УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Димитрије Марковић

МОДУЛАРНИ СИСТЕМ ЗА ОБРАДУ И СИГНАЛИЗАЦИЈУ АЛАРМНИХ ДОГАЂАЈА НА SMART HOME УРЕЂАЈИМА У EMBEDDED LINUX ОКРУЖЕЊУ

мастер рад

Београд, 2026.

Ментор:

др Мирко Спасић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

др Александар Картељ, ванредни професор
Универзитет у Београду, Математички факултет

др Мирослав Марић, редовни професор
Универзитет у Београду, Математички факултет

Датум одбране: _____

Наслов мастер рада: Модуларни систем за обраду и сигнализацију алармних догађаја на Smart Home уређајима у embedded Linux окружењу

Резиме: Уређаји за детекцију и сигнализацију опасности у стамбеним објектима морају обезбедити предвидљиво понашање и у условима отказа појединих компоненти, при чему неадекватна обрада непознатих или неисправних улазних података може представљати безбедносни ризик. У овом раду представљена је архитектура система за обраду и сигнализацију алармних догађаја на уређају са уграђеним рачунаром, чија је функционалност проверена на стварном хардверу. Систем је организован као скуп независних процеса који међусобно комуницирају искључиво разменом порука, при чему централни сервис доноси одлуке о потребним реакцијама, док три реактивна сервиса извршавају одговарајуће активности над сиреном, сигналном диодом и системом за слање обавештења другим уређајима. Компоненте система пројектоване су тако да енкапсулирају појединачне одлуке подложне променама, чиме се омогућава њихова независна измена без утицаја на остатак система. Улазни догађаји се валидирају пре покретања било какве реакције, а непознати типови алармних догађаја се намерно игноришу. Реакције на догађаје дефинисане су у табели акција, што омогућава увођење нових реакција без измене компоненте задужене за пријем и обраду догађаја. Функционалност система проверена је аутоматизованим тестовима заснованим на посматрању последица обраде догађаја, изведеним на циљној развојној плочи у више понављања, при чему су сви тестови успешно завршени и за сваки прихваћени алармни догађај потврђено је покретање све три предвиђене реакције. Резултати су показали и да се сервис за слање обавештења успешно опоравља након привремене недоступности комуникационог посредника, чиме је потврђено да предложена архитектура обезбеђује не само проширивост на нивоу имплементације већ и одговарајуће понашање система током извршавања.

Кључне речи: уграђени системи, модуларна архитектура, обрада вођена догађајима, међупроцесна комуникација, валидација улазних догађаја, алармни системи, поузданост, проширивост

Садржај

Глава 1

Увод

Уређаји са уграђеним рачунаром (енг. *embedded system*) одавно нису самостални. У савременом стану често су присутне десетине таквих уређаја, међусобно повезаних, а међу њима су и они чији је посао да упозоре на опасност, а то су детектори дима, сензори покрета, тастери за позив у помоћ и други. Од таквих уређаја не очекује се само да раде, него да раде предвидљиво и у случају отказа других уређаја и система. Једна од дефиниција уграђених система каже да су то рачунарски системи са вишим захтевима у погледу квалитета и поузданости него друге врсте рачунарских система [9], а ти захтеви су највиши управо код уређаја који упозоравају на опасност.

Проблем којим се овај рад бави јесте обрада и сигнализација алармних догађаја на једном таквом уређају. Улазну страну алармног система чине сензори и детектори, од детектора дима, топлоте, пламена и угљен-моноксида, преко инфрацрвених сензора покрета и контаката на вратима и прозорима, до детектора излива воде, надзорних камера и ручних тастера за позив у помоћ. На излазној страни стоје извршни органи (енг. *actuators*), у које поред сирене и светлосне сигнализације спадају и системи за одвођење дима, уређаји за аутоматско гашење пожара, електромагнетне браве које ослобађају путеве евакуације и вентили који затварају довод гаса. Обрада која те две стране повезује, дакле одлучивање о томе коју реакцију опажени догађај треба да покрене, предмет је овог рада.

Догађај настаје негде у систему, стиже до дела који одлучује шта треба предузети, и најчешће завршава као видљива или чујна реакција. Извори догађаја и сензори могу се додавати и мењати, начини сигнализације се временом проширују увођењем нових реакција, док правила одлучивања обично

остају релативно стабилна, иако се и она могу прилагођавати новим захтевима. Ако су пријем, одлучивање и реакција смештени у исти програм, свака измена у једном делу може да захтева нове у другим деловима система.

Из тога следи и мотивација за модуларну архитектуру. Систем у ком нови начин сигнализације захтева измену дела који прима и обрађује догађаје теже се проширује и теже испитује. Поред тога, алармни систем има особину коју обичне апликације немају: погрешна реакција на неразумљив или неисправан улаз може да произведе стварну безбедносну последицу. Лажна узбуна у првом тренутку може непотребно да узнемири кориснике и изазове погрешну реакцију, док често понављање таквих узбуна временом може да доведе до губитка поверења у систем и до тога да корисници престану да реагују на праве аларме.

У литератури постоји чврст оквир за оба питања (за модуларну поделу и за обраду вођену догађајима), али не и за конкретан механизам који се на овој класи уређаја користи за међупроцесну комуникацију. Механизам `ubus`, настао у оквиру пројекта OpenWrt, описан је првенствено инжењерском документацијом [11]. Зато се теоријски оквир у овом раду узима из два општија принципа, из поделе система на модуле и из размене порука по моделу објави/претплати се (енг. publish/subscribe). Сам механизам се затим посматра као један начин да се ти принципи остваре на уређају.

Циљ рада је да предложи и образложи архитектуру система за обраду и сигнализацију алармних догађаја, и да је провери на стварном хардверу. Предложено решење дели систем на независне процесе који комуницирају искључиво порукама. Централни сервис одлучује шта треба предузети, а реактивни сервиси ту одлуку спроводе одговарајућим акцијама. У овом раду су као пример описана три таква сервиса. То су сервис који управља сиреном, сервис који управља сигналном диодом и сервис који објављује обавештење другим уређајима.

Допринос рада је архитектура алармног система која истовремено остаје лако проширива и предвидљива при отказу, а прилагођена је ограничењима уграђеног Линукс окружења. Нова врста сигнализације уводи се без измене дела који прима и обрађује догађаје, а непознат или неисправан улаз не покреће ниједну реакцију.

Систем препознаје пет врста аларма, а то су пожар, провала, паника, излив воде и медицински позив. Свакој је додељен сопствени образац оглашавања,

дакле унапред задат ритам оглашавања сирене, као и боја и ритам сигналне диоде, по коме се врста аларма распознаје. На пример, пожар се оглашава ритмом од три кратка тона раздвојена паузама, док сигнална диода при томе брзо трепери црвено. Улазни догађаји се пре било какве реакције проверавају, а исход провере се разврстава у три класе. Реакције нису уграђене у обраду догађаја него уписане у табелу, тако да се нова реакција уводи без измене дела који прима догађаје. Понашање целине проверава се аутоматизованим алатом који, пошто алармни догађај не враћа одговор, испитује последицу уместо повратне вредности.

Рад је организован тако да глава 2 уводи појмове на које се остатак ослања, глава 3 излаже и образлаже архитектуру, а глава 4 проверу улазних догађаја. Глава 5 описује практичну имплементацију, глава 6 проширивост система, глава 7 тестирање и његове резултате, а глава 8 закључује рад.

Глава 2

Преглед области и теоријски оквир

Појмови на које се ослањају одлуке из наредних глава долазе из четири области: архитектуре уграђених система, поузданости безбедносно значајних система, међупроцесне комуникације и модуларне декомпозиције. Свака је овде изложена у обиму у ком се касније заиста користи.

2.1 Уређаји са уграђеним рачунаром

Уређај са уграђеним рачунаром нема једну општеприхваћену дефиницију. Уобичајених описа има више и ниједан не покрива целу породицу таквих уређаја, јер се граница помера са сваким напретком технологије и падом цене компоненти [9]. Један од тих описа је за овај рад значајнији од осталих. Уграђени системи су рачунарски системи са вишим захтевима у погледу квалитета и поузданости него друге врсте рачунарских система, при чему се распон креће од уређаја код којих је квар непријатност до оних код којих представља опасност по живот [9]. Алармни уређаји у стану припадају ближе другом крају тог распона него првом.

Структура таквих уређаја описује се моделом са три слоја: хардверски слој, слој системског софтвера и слој апликације, при чему је само хардверски обавезан, а преостала два нису [9]. Систем описан у овом раду има сва три: плочу са излазима за сирену и диоду, оперативни систем са системским библиотекама, и три сопствена сервиса у слоју апликације.

Прототип је развијан и испитан на развојној плочи са процесором NXP

i.MX93, на којој ради оперативни систем Линукс изграђен алатом *Yocto*[1] [15], са *systemd* као системом за управљање сервисима. Од хардвера који овај рад користи, плоча има излазе за сирену, статусну диоду са три канала (црвени, зелени и плави, чијим се комбиновањем добија боја којом диода светли) и екран са подесивим позадинским осветљењем, а оперативни систем нуди приступ њима кроз систем датотека `sysfs`, где се сваки излаз види као засебна датотека у коју се уписује вредност. Корени систем датотека монтиран је само за читање, док за податке који треба да преживе поновно покретање постоји засебна партиција. Плоча има и мрежну везу.

Архитектура уграђеног система дефинише се као апстракција уређаја, дакле као приказ у ком су детаљи имплементације изостављени, а описано је само његово понашање [9]. Тако схваћена, архитектура је оно што се може разматрати и оспоравати пре него што иједна линија кода постоји, и управо је то предмет главе 3.

Шири контекст у ком овај уређај ради јесте паметна кућа (енг. *Smart Home*), где више уређаја различитих произвођача дели простор и међусобно утиче једни на друге. Та особина је важна за наредни одељак, јер помера тежиште са исправности појединачног уређаја на понашање целине.

2.2 Поузданост и безбедност алармног система

Реч „безбедност” у контексту повезаних уређаја има два различита значења која се лако могу помешати. Једно је заштита од злонамерног упада, друго је спречавање да уређај доведе до штете својим редовним радом или отказом. У овом раду фокус је на другом значењу.

У паметним кућама заснованим на интернету ствари (енг. *Internet of Things*) ти ризици се посматрају независно од заштите од упада, пошто лоше међудејство повезаних уређаја, механички отказ уређаја и неуспела комуникација могу увести систем у непоуздана и опасна физичка стања [1]. Пример тога је отказ детектора гаса уз укључен шпорет - опасност која

[1] *Yocto* је скуп алата за прављење сопствене дистрибуције Линукса за уређаје са уграђеним рачунаром. Уместо готове дистрибуције, из рецепата се гради слика система прилагођена конкретној плочи.

настаје без иједног нападача. Управо ту врсту отказа алармни систем треба да открије и да на њу сигнализира.

Формални оквири за такве системе постоје. На пример, стандард *IEC 61508* је генерички стандард функционалне безбедности за електричне, електронске и програмабилне електронске системе повезане са безбедношћу, писан тако да се може користити непосредно, али и као основа за секторске и производне стандарде. Из њега су изведени, између осталих, стандарди за машине, процесну индустрију, нуклеарна постројења и погоне [8]. Појам нивоа интегритета безбедности уведен је због систематских отказа, дакле отказа који потичу из спецификације и пројекта, а не из хабања компоненти [8].

Прототип описан у овом раду није развијан по том стандарду, али је из њега преузет начин расуђивања, то јест да се понашање при отказу и при неочекиваном улазу пројектује унапред, а не препушта случају.

На нивоу софтвера тај начин расуђивања има свој каталог. Међу обрасцима пројектовања за безбедносно-критичне уграђене системе безбедно стање (енг. *fail-safe state*) дефинисано је као стање које се, у случају отказа, може препознати као безбедно и без ризика [2]. Тај појам је веома важан и представља основу одлуке описане у одељку 4.3.

Поред кровног стандарда постоји и стандард писан баш за ову врсту уређаја. *EN 50131 - Alarm systems - Intrusion and hold-up systems* објављен је у више делова. Први део прописује системске захтеве за системе за дојаву провале и препада, а кроз све делове важе четири степена безбедности [6]. Пун текст није коришћен у изради овог рада, па се стандард овде наводи као оквир, без позивања на појединачне захтеве по степену.

2.3 Међупроцесна комуникација и модел објави/претплати се

Када је систем подељен на више процеса, начин на који они комуницирају је јако важан и представља део архитектуре. Ако процеси деле меморију, размена је брза, али грешка у једном процесу може да поквари податке другом. Ако размењују поруке, размена кошта нешто више, али сваки процес држи своје стање само за себе.

Међу моделима размене порука, за системе који реагују на догађаје најзначајнији је модел објави/претплати се. У њему претплатници изражавају своју

заинтересованост за догађај или образац догађаја и потом бивају асинхроно обавештени о догађајима које производе издавачи. Главна особина заједничка свим његовим варијантама је потпуна раздвојеност учесника у три димензије [5].

- **Просторна раздвојеност** - учесници не морају да знају једни за друге, нити колико их има.
- **Временска раздвојеност** - издавач и претплатник не морају да буду активни у истом тренутку.
- **Синхронизациона раздвојеност** - издавач се не блокира док производи догађај, а претплатник може бити обавештен независно од свог тренутног тока извршавања.

Догађај и наредба нису иста врста поруке, иако се обе преносе истим механизмом. Та разлика има и своја имена: *порука о догађају* (енг. *Event Message*) служи да се о нечему што се догодило обавести друга страна, при чему је сам садржај често мање важан од чињенице да је порука стигла, док *командна порука* (енг. *Command Message*) служи да једна апликација покрене функцију друге [7]. Систем описан у овом раду користи обе, и то намерно различито, што је образложено у одељку 3.4.

Конкретан механизам који све то преноси јесте `ubus`, настао у оквиру пројекта OpenWrt. Централну улогу има посреднички процес `ubusd`. Учесници се на њега повезују преко Unix сокета, а поруке се преносе у бинарном формату типа, дужине и вредности [11]. Претплата (енг. *subscribe*) не мора бити на један догађај, него може обухватити целу фамилију догађаја, тако што се уместо тачног имена наведе образац у ком звездица на крају стоји уместо било ког наставка имена. За овај рад је посебно значајно то што ubus подржава оба облика комуникације која су потребна систему: објаву догађаја и позив метода регистрованих објеката. Објава догађаја користи се када неки део система пријављује да се аларм догодио, док се позив метода користи када централни сервис треба да упути конкретну команду реактивном сервису. На тај начин се у истом механизму задржава јасна разлика између поруке која описује појаву догађаја и поруке која захтева извршавање радње.

Механизам `ubus` описан је првенствено инжењерском документацијом. Зато се теоријски оквир у овом раду узима из општијих принципа: из модела

објави/претплати се и из принципа модуларне декомпозиције. Сам `ubus` се затим посматра као један начин да се ти принципи остваре на самом уређају.

2.4 Принципи модуларне декомпозиције

Питање како поделити систем на модуле старије је од већине данашњих система и алата. Исти програм се може поделити на више различитих начина тако да му функционалност остане иста. Проблем настаје када дође до измена програма, где различите поделе воде ка различито компликованим изменама. Боља је она подела у којој сваки модул крије по једну одлуку која се временом може мењати [12]. Подела по корацима обраде делује природније јер прати редослед посла, али даје модуле који се при свакој измени морају мењати заједно.

Тај принцип је касније добио и практичан облик. Архитектонска тактика је пројектна одлука која утиче на то колико добро архитектура испуњава неки захтев квалитета, а тактике за лакшу измену своде се на мање спрезања међу модулима, више кохезије унутар модула и одлагање везивања одлука до тренутка када су заиста потребне [3]. Таква подела олакшава измене, али истовремено уводи додатну сложеност, јер границе између модула, њихове интерфејсе и начин комуникације треба јасно пројектовати и касније одржавати. Захтеви квалитета се при томе не бирају независно један од другог. Олакшавање каснијег увођења измена, као и побољшање расположивости, брзине или безбедности, најчешће иде на штету неког од преосталих захтева [4].

Оно што ниједан од радова наведених у овој глави не решава јесте питање када је таква подела оправдана у конкретном систему. Постоји и приступ у ком се корист од поделе на модуле процењује техникама вредновања опција, проверен на Парнасовим сопственим примерима [14]. За рад ове врсте то је подсетник да свака граница између модула има своју цену, па се подела не уводи сама по себи, већ онда када корист коју доноси надмашује сложеност коју уводи. У овом раду та корист се огледа у томе што се пријем и класификација алармних догађаја раздвајају од конкретних начина реакције. Таква подела омогућава да се нови начин сигнализације, као што је нови локални сервис или мрежно обавештавање, дода без измене основне логике за обраду алармних догађаја. На исти начин се уводе и нове врсте аларма, дакле поруке које шаљу накнадно додати сензори, детектори и други уређаји.

Глава 3

Архитектура система

Систем је подељен на четири процеса која међусобно комуницирају искључиво разменом порука. Ова глава образлаже зашто је подела повучена баш тако, шта сваки процес зна, шта намерно не зна и које су алтернативе разматране и одбачене. Фокус није на начину на који је нешто имплементирано, него на месту на коме је постављена граница између компоненти.

3.1 Преглед архитектуре и ток алармног догађаја

Систем чине четири засебна процеса. Централни сервис `emergency-notifyd` прима алармне догађаје и одлучује шта треба предузети. Три реактивна сервиса извршавају ту одлуку.

- `emergency-siren-service` - управља сиреном.
- `emergency-led-service` - управља статусном сигналном диодом.
- `emergency-mqtt-service` - објављује обавештење о алармном догађају другим уређајима.

Прва два делују на локални хардвер, док трећи шаље податак са уређаја, и то протоколом MQTT, лаким протоколом по моделу објави/претплати се, намењеним уређајима са ограниченим ресурсима [10]. Та разлика је важнија него што на први поглед изгледа и о њој је реч у глави 6.

Уз њих постоји и `emergency-notify-sim`, алат за аутоматизовано тестирање. Он није део радног система и на уређају се не покреће, али користи исти комуникациони пут као сервиси, то јест шаље алармне догађаје и очитава стање реактивних сервиса, те се помиње свуда где је реч о провери понашања система.

Ниједан од ова четири процеса не дели меморију са осталима. Целокупна комуникација одвија се преко механизма `ubus`, лаког система за међупроцесну комуникацију (енг. *inter-process communication*) развијеног у оквиру пројекта OpenWrt [11]. Централну улогу има посреднички процес `ubusd`, преко кога се поруке усмеравају. Учесници комуницирају преко Unix сокета, а поруке се преносе у формату типа, дужине и вредности (енг. *type-length-value*). Подела на процесе резултује тиме да се сваки сервис може зауставити, изменити и поново покренути без поновног превођења осталих.

Табела 3.1: Процеси система и њихове одговорности

Процес	`ubus` објекат	Одговорност
`emergency-notifyd`	`emergency.notify`	одлучивање о реакцији
`emergency-siren-service`	`emergency.siren`	управљање сиреном
`emergency-led-service`	`emergency.led`	управљање сигналном диодом
`emergency-mqtt-service`	`emergency.mqtt`	објављивање обавештења

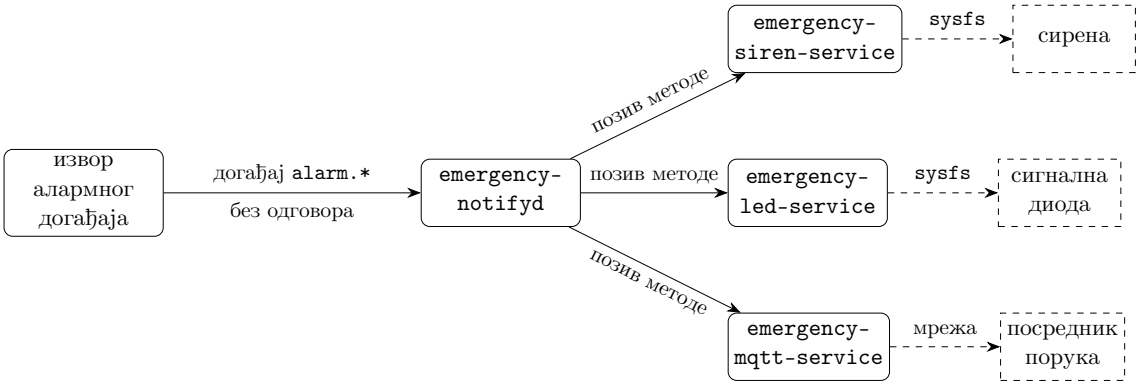
Табела 3.1 показује поделу одговорности и називе `ubus` објеката које сервиси региструју. Алат за тестирање у њој не стоји зато што не региструје сопствени објекат, то јест он ништа не нуди осталим процесима, већ само шаље догађаје и очитава стање реактивних сервиса.

Ток једног алармног догађаја има четири корака. Произвођач догађаја, у стварном систему сензор, детектор, камера или ручни тастер, а у тестном окружењу `emergency-notify-sim`, шаље догађај чије име почиње са `alarm.` и завршава се типом аларма, на пример `alarm.fire`, заједно са пратећим подацима. Сервис `emergency-notifyd`, који је регистровао ослушкивач за образац `alarm.*`, дакле за целу ту фамилију, прима тај догађај и проверава исправност пратећих података. Ако су подаци исправни, редом позива регистроване акције. Свака од њих сама одлучује да ли препознаје тај тип аларма. Акција за сирену тада позива методу `alarm` над објектом `emergency.siren` и прослеђује јој идентификатор сирене, трајање и обележје обрасца оглашавања (дакле начина на који се сирена оглашава, ритма смењивања тонова по ко-

ме се врста аларма распознаје), акција за диоду поступа исто над објектом `emergency.led`, а акција за обавештење над објектом `emergency.mqtt`. Сваки реактивни сервис затим изводи своју реакцију.

У овом току важно је разликовати две врсте комуникације. Прва је објава алармног догађаја, којом се систем обавештава да се нешто догодило и која не враћа одговор пошиљаоцу. Друга је позив методе над регистрованим `ubus` објектом, којим централни сервис од реактивног сервиса тражи извршавање конкретне радње и добија одговор о исходу позива. Начин на који централни сервис бира акције описан је у одељку 3.3.

Тај ток приказан је на слици 3.1. Пуне стрелице означавају пренос преко механизма `ubus`, а испрекидане оно што сваки реактивни сервис ради даље. Два сервиса непосредно приступају хардверу, док трећи шаље поруку посреднику.



Слика 3.1: Ток алармног догађаја од извора до хардвера

На слици се види и асиметрија о којој је реч у одељку 3.4. Лева стрелица представља догађај који не враћа одговор, док десне представљају позиве методе који одговор враћају.

3.2 Декомпозиција по критеријуму сакривања информација

Подела система на модуле може се извести по више различитих критеријума, а избор критеријума одређује колико ће систем бити подложан изменама. Подела по корацима обраде даје модуле код којих свака измена захтева измене у више модула, док подела по томе *шта модул скрива* даје модуле који

се могу мењати независно [12]. Модул по том схватању постоји да сакрије једну пројектну одлуку која се временом може променити, и да ту одлуку не пропусти кроз свој интерфејс.

Ова архитектура следи тај критеријум у више модула. Сервис `emergency-notifyd` скрива пресликавање типа аларма у параметре реакције. Он зна да пожар треба да произведе одређени образац оглашавања, али не зна ниједан детаљ о томе како се тај образац остварује. То пресликавање није записано као табела него као функција `resolve_siren_action_config()` у модулу `siren_action.c`, која поређењем имена догађаја враћа структуру `siren_action_config` са пољима `siren_id`, `duration_sec` и `pattern_name`. За тип који не препознаје, функција враћа структуру са пољем `valid` постављеним на нулу.

Супротну страну исте границе скривају хардверски модули реактивних сервиса. Модул `siren_hw.c` је једини део система који познаје `sysfs` путање сирене, бројеве излаза и вредности периода и радног циклуса. Модул `led_hw.c` на исти начин затвара путање сигналне диоде. Ово није један изузетак него исти принцип примењен двапут, што је уједно и провера да принцип није накнадно измишљен ради објашњења (други реактивни сервис настао је касније, по узору на први, и границу је повукао на истом месту). Слој за логовање скрива трећу променљиву одлуку, избор библиотеке која заиста уписује логове, тако да њена замена не утиче ни на један сервис.

Четврта примена истог принципа не тиче се ни хардвера ни логовања. Модул `mqtt_client.c` у сервису за обавештавање скрива библиотеку преко које се поруке шаљу, тачно онако како `siren_hw.c` скрива приступ хардверу. Слој изнад њега зна да треба објавити поруку на одређену тему, али не зна ниједан детаљ о томе како се веза успоставља ни коју библиотеку систем користи. То је потврда да критеријум поделе није везан за хардвер него за променљивост. Све четири границе постоје зато што иза сваке стоји одлука која се временом може променити.

Иста граница повучена је и унутар самог сервиса `emergency-notifyd`. Модул `event_validation.c` проверава искључиво структуру пратећих података, дакле да ли су обавезна поља присутна, да ли је вредност поља `severity`, односно степен критичности аларма, у дозвољеном скупу вредности и да ли се поље `type` слаже са именом примљеног догађаја. Списак подржаних типова аларма он не познаје. Тај списак се налази у самим акцијама, у `siren_`

`action.c` и `led_action.c`, зато што су оне једине којима је потребан. Да је списак поновљен и у валидацији, додавање новог типа аларма захтевало би усклађену измену оба модула, а свака таква измена носи ризик да се два списка временом разиђу.

Из те одлуке следи друга. Ако само акција зна које типове препознаје, онда она мора и да јави да ли је примљени тип био међу њима. Зато акције не враћају целобројну ознаку успеха него вредност набројивог типа `action_result_t`, издвојеног у заглавље `action_result.h`, са три исхода: догађај је прихваћен, догађај је игнорисан као неподржан, или је дошло до грешке при извршавању. Тип је намерно заједнички за све акције. Да свака акција има сопствени набројиви тип, централни сервис не би могао да их обрађује на исти начин, а управо то и омогућава регистар описан у одељку 3.3. Без три различите вредности позивалац не би могао да разликује неподржан тип аларма од неуспеле акције, а те две ситуације имају различите узроке и траже различит одговор.

Разматрана је и очигледнија подела, у којој би `emergency-notifyd` сам управљао сиреном. Она је одбачена. У таквој подели централни сервис би морао да познаје `sysfs` путање и бројеве излаза, па би свака измена хардвера (други модел сирене, друга плоча, друга нумерација излаза) захтевала измену сервиса чији посао са хардвером нема везе. Одвојен реактивни сервис ту измену затвара унутар модула `siren_hw.c`. Колико та одбачена подела кошта видело се тек када је додат сервис за сигналну диоду. Да је хардвером управљао централни сервис, свака нова врста сигнализације тражила би нов хардверски код баш у њему.

Избор механизма за истовремено извршавање такође прати поделу одговорности, а не једно правило за све. Сервис `emergency-notifyd` је једнонитан (енг. *single-threaded*), па се и обрада догађаја и одговор на упит о стању извршавају редом у истој петљи догађаја. Структура са бројачима `g_stats` у модулу `notify_stats.c` зато нема никакву заштиту од истовременог приступа. Осим ње, једино променљиво стање на нивоу датотеке у овом сервису је показивач `ctx` на `ubus` контекст у `notifyd_ubus.c`, а остале структуре се после регистрације не мењају. Реактивни сервиси су другачији, јер образац оглашавања траје у времену и мора да се извршава независно од пристиглих позива. Сервис `emergency-siren-service` зато има радну нит (енг. *thread*) и штити дељено стање механизмом узајамног искључивања `pattern_lock`. Увођење истог механизма и у `emergency-notifyd` додало би сложеност и навело

читаоца кода на погрешан закључак да се нешто извршава паралелно.

3.3 Регистар акција

У току развоја, док је постојао само један реактивни сервис, начин на који га централни сервис позива није изгледао као нешто о чему треба одлучивати. Сервис `emergency-notifyd` звао је обраду сирене непосредно, и то је изгледало довољно. Увођењем другог реактивног сервиса то више не важи: непосредан позив би значио да свако додавање нове врсте сигнализације тражи измену обраде догађаја у централном сервису, дакле измену кода који са том сигнализацијом нема никакве везе.

Уместо тога уведен је регистар акција. Тип `alarm_action_t` у заглављу `alarm_action.h` садржи име акције и показивач на функцију која је обрађује, а сам регистар је статичка табела `alarm_actions` у модулу `alarm_action.c`, приказана на листингу 3.1.

```
1 const alarm_action_t alarm_actions[] = {
2     { "siren", siren_action_handle_alarm_event },
3     { "led",   led_action_handle_alarm_event },
4     { "mqtt",  mqtt_action_handle_alarm_event },
5 };
```

Листинг 3.1: Регистар акција са три уписане реакције

Обрада догађаја у `notifyd_ubus.c` више не помиње ниједну појединачну акцију. Она пролази кроз табелу и сваку регистровану акцију позива са истим аргументима, ослањајући се на то да све враћају исти тип исхода. Табела тренутно има три реда. Трећи је додат пошто су прва два већ радила, а додавање се свело на два корака: нов ред у табели и нову датотеку у којој је написана сама обрада те акције. Датотека `notifyd_ubus.c`, у којој се догађаји примају, није при томе измењена. Обим посла при додавању нове реакције тиме не зависи од тога колико их систем већ има, јер обрада догађаја остаје иста без обзира на број уписаних акција.

Прелазак са једне акције на више њих отворио је питање које раније није постојало: како се класификује догађај када акција има више, а не слажу се све у одговору. Усвојено правило је да догађај буде прихваћен ако га је бар једна акција препознала, а игнорисан ако га ниједна није препознала. Отказ извршавања не мења ту оцену. Ако акција препозна тип аларма али

позив ка реактивном сервису не успе, догађај се и даље броји као прихваћен, а отказ се бележи посебно. Класификација описује догађај, док је успешност акције засебна димензија. Мешањем то двоје изгубила би се разлика између неисправног улаза и исправног улаза на који хардвер није одговорио, а то су различити кварови са различитим узроцима.

3.4 Догађаји и удаљени позиви

Кроз систем пролазе две врсте порука, а разликују се по томе да ли пошиљалац добија одговор. Алармни догађај облика `alarm.*` шаље се без очекивања одговора. Пошиљалац га објави и настави свој посао. Позив методе `alarm` над објектом `emergency.siren` или `emergency.led` понаша се супротно. Позивалац чека и добија исход. Ова разлика није случајна, већ смислено осмишљена, и свака страна има свој разлог.

За алармне догађаје изабран је модел објави/претплати се, а за позиве ка реактивним сервисима командна порука. Оба појма и разлика међу њима изложени су у одељку 2.3, а овде је реч о томе зашто је баш таква подела примењена на овај систем.

Разлог за такав избор долази из природе алармног система. Извор аларма (сензор, тастер за панику или, у тестном окружењу, симулатор) нема разлога да зна колико реактивних сервиса постоји, нити који су. Његов посао престаје у тренутку када објави да се нешто догодило. Да је уместо тога изабран непосредан позив, сваки нов начин сигнализације захтевао би измену на страни извора аларма, што је управо спрега коју подела на процесе треба да уклони.

Позиви ка реактивним сервисима су намерно другачији. Централни сервис мора да зна да ли је акција извршена, јер отказ хардвера или недоступан сервис нису исто што и уредно реализована реакција. Та разлика се броји одвојено и о њој је реч у глави 4. Командна порука која не захтева одговор ту информацију не би могла да пренесе.

Из природе догађаја који не враћа одговор следи још једна последица, овог пута релевантна за тестирање. Пошиљалац алармног догађаја нема од кога да добије потврду да је догађај обрађен, па се исправност обраде не може проверити одговором. Аутоматизовани тест зато мора да провери *последицу*, стање реактивног сервиса и вредности бројача, а не повратну вредност слања. Начин на који то ради описан је у глави 7.

3.5 Апстракција хардвера

Слојевити модел уграђених система раздваја хардверски слој, слој системског софтвера и слој апликације. Најнижи део системског софтвера чине управљачки програми уређаја (енг. *device drivers*), односно софтверске библиотеке које припремају хардвер и уређују приступ хардверу са виших слојева софтвера, и које су спона између хардвера и оперативног система, међуслоја и апликације [9]. Иста књига наводи и да оперативни систем од таквог кода често тражи одређен скуп функција: покретање, гашење, омогућавање и онемогућавање [9].

Управо тај скуп препознаје се у оба сервиса која управљају хардвером, где је слој апстракције једна датотека. Заглавље `siren_hw.h` објављује четири операције: припрему, ослобађање, укључивање и искључивање сирене. Ништа више од тога није доступно вишим слојевима, ни путања, ни број излаза, ни вредности периода и радног циклуса. Заглавље `led_hw.h` објављује нешто богатији скуп, зато што је и уређај богатији: поред припреме и гашења свих канала, ту је и постављање јачине појединачног канала, где је канал једна од три вредности набројивог типа за црвену, зелену и плаву компоненту.

Разлика између та два интерфејса последица је разлике међу самим уређајима. Сирена се у овом систему понаша као бинарни уређај, па њен интерфејс има облик прекидача. Сигнална диода има боју и јачину, па њен интерфејс има параметре. Апстракција није преписана са једног уређаја на други, него је сваки интерфејс сведен на операције које тај уређај заиста захтева. Оно што је заједничко није облик интерфејса него правило да све што има везе са `sysfs` остаје у `siren_hw.c`, односно у `led_hw.c`, а слој изнад ради са појмовима уређаја.

Једна операција у интерфејсу диоде заслужује посебну пажњу, јер у себи носи безбедносну одлуку. Уређај има екран са подесивим позадинским осветљењем, а функција `led_hw_ambient_brightness()` јачину сигналне диоде везује за затечену јачину тог осветљења. Док је екран осветљен пуном јачином, диода светли пуном, а када је екран пригушен, пригушена је и диода. Када путању за очитавање затеченог осветљења није могуће прочитати, она враћа пуну јачину, а не пригушену. Разлог је исти онај који се провлачи кроз цео систем. Када уређај не може да утврди у ком је стању окружење, бира понашање које не умањује видљивост аларма. Ова функција такође

никада не пријављује грешку позиваоцу, него увек враћа употребљиву вредност, чиме немогућност очитавања не може да прекине сигнализацију. Путања са које се то осветљење очитава записана је као једна именована константа, `/sys/class/backlight/backlight-lcd/brightness`. Провером на цильној плочи потврђено је да тамо постоји тачно један уређај за осветљење екрана, изложен под тим именом, и да је та путања исправна.

Граница по којој се бира између дневног и ноћног режима стоји у коду као именована константа, а не као број усред израза. Опсег вредности на уређају иде од нуле до сто, па је граница постављена на средину распона. Ако се на другом уређају опсег разликује, помера се само та једна константа.

3.6 Апстракција логовања

Исти поступак сакривања променљиве одлуке иза границе, примењен у претходном одељку за приступ хардверу, примењен је и на логовање. Сервиси не позивају ниједну библиотеку за логовање непосредно, него слој `emergency_log_*`, скуп функција из заглавља `emergency/log.h`. Тај слој познаје четири нивоа логова и име сервиса, које се прослеђује при покретању и постаје категорија логова. Све остало (где логови иду, у ком формату и да ли се датотеке логова ротирају[1]) решава се у самој имплементацији и сервиси о томе не знају ништа.

Тај интерфејс има две имплементације. Једна уписује на стандардни излаз, друга користи библиотеку *zlog*. Избор се доноси у време превођења, преко опције `ENABLE_ZLOG` у систему превођења кода, а не у време извршавања. То је свесно ограничење, јер се о начину логовања одлучује приликом испоруке, а не током рада уређаја.

Тај избор могао је бити и подешавање у датотеци, које би се мењало без поновног превођења. Од тога се одустало зато што избор у време превођења омогућава да се систем преведе и на рачунару за развој, где библиотека *zlog* не мора бити ни присутна.

Библиотека *zlog* изабрана је зато што њен начин подешавања одговара подели на више процеса. Логови се разврставају по категоријама, а правило повезује категорију, ниво, одредиште и формат [13]. За систем са више процеса

[1]Ротација подразумева да се текућа датотека, када пређе задату величину, преименује, а писање наставља у новој. Чува се одређен број старијих датотека, док се најстарије бришу, чиме логови престају да расту неограничено.

важно је и то што библиотека безбедно ротира датотеке и са више процеса и са више нити, при чему за нити користи мутекс, а за процесе саветодавно закључавање записа (енг. advisory locking)[2] системским позивом `fcntl`. Овде четири сервиса раде истовремено, а уз њих повремено и алат за тестирање.

Понашање при отказу је, међутим, важније од избора библиотеке. Ако покретање *zlog* имплементације не успе (јер нема подешавања, нема права уписа или нема директоријума), систем исписује упозорење и прелази се на испис на стандардном излазу, па наставља рад. Чак и када слој за логовање никада није ни покренут, поруке се и даље исписују уместо да буду тихо одбачене. Алармни сервис не сме престати да ради зато што логовање не ради. Логовање је помоћна функција, а обрада аларма основна. Тиме је и овде примењен образац безбедног стања, јер систем при отказу прелази у унапред предвиђено стање чије су последице познате и безопасне [2].

Свака апстракција има и цену, а ова своју показује баш у ономе што скрива. Библиотека за логовање уме уз сваки лог да забележи датотеку и ред из ког је позвана, али кроз овај слој пролазе само ниво записа, име сервиса и сама порука. Пошто библиотека види једино позив из омотача, забележено место увек показује на исти ред унутар слоја. Место одакле је порука заиста потекла зато мора бити видљиво из њеног текста, што је и разлог што поруке у овом систему носе име догађаја, објекта или акције. Слој који сакрива библиотеку од сервиса истовремено сакрива и идентитет позиваоца од библиотеке.

Одредиште логова захтевало је посебну пажњу. На системима грађеним алатом *Yocto* директоријум `/var/log` често је симболичка веза ка привременом систему датотека у радној меморији, што значи да логови тамо не преживљавају поновно покретање, а то поништава главни разлог за сопствену инфраструктуру логовања. На циљној плочи је провером утврђено да важи управо то, и уз то да је корен система датотека монтиран само за читање. Логови зато иду на партицију која преживљава поновно покретање, а системска јединица (енг. *systemd unit*) чека да се та партиција монтира пре него што почне да пише. Предност овог решења је у томе што одредиште бира апликација, а не подешавање слике оперативног система (енг. *image*).

[2]Саветодавно закључавање подразумева да оперативни систем не спречава упис ономе ко браву не провери. Договор важи међу процесима који га поштују, што је овде случај јер сви користе исту библиотеку.

3.7 Разматране алтернативе

Одлуке добијају значење тек уз оно што је уз њих одбачено. Неке од тих алтернатива већ су образложене на местима где су настале (подела у којој централни сервис сам управља сиреном у одељку 3.2, подразумевани образац за непознат аларм у глави 4), па се овде не понављају. Остале разматране алтернативе наведене су у наставку.

Најједноставнија замислива подела је да поделе нема, то јест један процес који све ради. Она је одбачена пре него што је писање кода почело. Такав систем се теже проширује, јер свака нова врста сигнализације улази у исти извршни програм. Теже се и тестира, јер се појединачни део не може покренути сам. Не дозвољава се да један део буде заустављен и замењен док остали раде. Тренутна подела на четири процеса ту цену плаћа унапред, у виду међупроцесне комуникације која у монолиту не би постојала.

Прва скица реактивног сервиса била је имитација (енг. *mock*), дакле сервис који само памти да ли је сирена укључена, без интеграције хардвера. Одбачена је намерно. Таква имитација би олакшала тестирање, али би читав слој апстракције хардвера остао непроверен, а управо је он место где се показује да подела има смисла. Систем који никада није управљао стварним уређајем не може да тврди ништа о примењивости.

Са увођењем сервиса за сигналну диоду отворила су се два питања. Прво је на шта тај сервис сме да има утицај. Одлучено је да управља само статусном диодом, а не и диодом напајања, зато што је он реакција на аларм, а не управљање изгледом уређаја. Друго отворено питање је било које методе такав сервис треба да понуди. Разматрано је да се цело стање диоде поставља једним позивом, али је одбачено у корист истог скупа метода који већ има сирена: `alarm` за покретање реакције, `cancel_alarm` за њено отказивање и `status` за очитавање стања. Тек тако централни сервис може све реактивне сервисе да зове на исти начин, што је и претпоставка регистра из одељка 3.3.

Избор библиотеке за логовање имао је три кандидата. Позната библиотека *spdlog* одбачена је зато што је писана за језик *C++*, док су сервиси у овом систему написани у језику *C*. Системски алат *logrotate* решава само ротацију датотека, а не и разврставање логова по сервису, што је овде био главни захтев. Изабран је *zlog*, из разлога наведених у одељку 3.6.

Глава 4

Валидација и класификација улазних догађаја

Алармни систем прима податке од уређаја и програма над којима нема контролу, па мора да одлучи шта ће са поруком коју не разуме. Од такве одлуке зависи да ли ће непотпуна или противречна пријава покренути сирену, а погрешан избор ту није само програмска грешка него безбедносни проблем. Ова глава описује како се одлука доноси, како се исход именује и како се броји.

4.1 Модел валидације структуре догађаја

Уз име догађаја стиже и пратећи садржај у бинарном облику који `ubus` користи за преношење структурираних података. Модул `event_validation.c` тај садржај најпре рашчлањује према унапред задатој политици (енг. policy), у којој су наведена четири поља (`type`, `severity`, `source` и `message`), сва четири као ниске знакова. Поља која нису у политици не стижу до остатка обраде.

Прва три поља су обавезна, а `message` није. Разлика је намерна. Прва три носе податке на основу којих се одлучује, док је порука намењена човеку који касније чита логове. Одсуство обавезног поља и потпуно празан садржај воде до два различита исхода. Празан садржај значи да пошиљалац није послао ништа, док садржај у коме постоји само поље `message` значи да јесте слао, али без података на основу којих се одлучује. То су два различита квара на страни пошиљаоца, и ова информација је важна за откривање грешке. Када би провера за оба квара враћала исти исход, из њега се не би могло закључити

који је од њих наступио.

Друге две провере тичу се вредности, не присуства. Поље `severity` мора бити једна од четири дозвољене вредности (`low`, `medium`, `high` или `critical`), а поље `type` мора се поклапати са делом имена догађаја иза последње тачке. Догађај `alarm.fire` са пољем `type` постављеним на `burglary` биће одбијен као неисправан. Такав догађај је знак да пошиљалац не функционише исправно. Пошто догађаји путују преко локалне магистрале унутар истог уређаја, оштећење поруке у преносу није вероватно, али јесте да пошиљалац име догађаја и поље `type` поставља на два места у коду, па после увођења новог типа једно остане неизмењено. У алармном систему је безбедније одбити противречан улаз него произвољно изабрати који је од два податка меродаван.

Исход провере није логичка вредност него једна од пет вредности набројивог типа `event_validation_status_t`.

- **Исправно** - садржај има сва обавезна поља и све дозвољене вредности.
- **Нема садржаја** - догађај је послат без иједног поља.
- **Недостаје обавезно поље** - на пример садржај без поља `source`.
- **Недозвољена озбиљност** - уместо једне од четири дозвољене вредности стигне, рецимо, `urgent`.
- **Неслагање типа са именом догађаја** - као у претходном примеру, где уз `alarm.fire` стиже тип `burglary`.

Провера није ослоњена ни на једну библиотеку, али прати исти модел који за структурну валидацију *JSON* докумената прописује обавезна својства и унапред набројан скуп дозвољених вредности [16]. Тај модел је описан над текстуалним *JSON* документима, где се поља добијају рашчлањивањем текста. Овде се исте провере примењују на бинарни облик у ком `ubus` преноси поруке, у ком су поља већ издвојена као парови имена и вредности, па се проверава њихово присуство и садржај, а не облик текста.

4.2 Класификација исхода обраде

Свака обрада алармног догађаја завршава се једном од три оцене, наведене у набројивом типу `alarm_event_classification_t`.

- **Неисправан** - догађај који није прошао проверу из одељка 4.1. Ту оцену доноси искључиво валидација и она не зависи ни од једног реактивног сервиса.
- **Игнорисан** - догађај који је структурно исправан, али чији тип ниједна регистрована акција не препознаје.
- **Прихваћен** - догађај који је бар једна акција препознала као свој.

Те три оцене нису произвољне, јер свака настаје на другом месту у обради и носи другачији податак о томе шта је са пријавом било.

Подела посла између провере и акција је оно што одржава систем проширивим. Валидација проверава облик и не зна ништа о томе који су типови аларма подржани, јер се тај списак налази у самим акцијама. Да валидација има сопствену листу подржаних типова, додавање новог типа тражило би усклађену измену на два места. Последица те поделе је да валидација не може да разликује непознат тип аларма од познатог. То може само акција, и зато акција мора да врати оцену, а не само ознаку успеха.

Отуда и трећа вредност у типу `action_result_t`. Акција може да јави да је догађај прихватила, да га није препознала, или да га јесте препознала али да извршење није успело. Само са прве две вредности систем не би могао да разликује неподржан тип аларма од недоступног реактивног сервиса, а то су различити кварови. Први долази од пошиљаоца, други од самог уређаја.

4.3 Безбедно подразумевано понашање

Прва замисао била је да непознат тип аларма покрене подразумевани образац оглашавања, по логици да је боље огласити се него не реаговати. Та замисао је одбачена и замењена супротном, па се непознат догађај игнорише и ништа се не покреће.

Разлог је безбедносни. Систем који на непознат улаз покреће сирену даје нападачу или неисправном уређају једноставан начин да изазове лажну узбуну, а низ лажних узбуна поуздано доводи до тога да људи престану да реагују на праву. Уз то, подразумевано понашање сакрива грешку, јер би тип који је требало да буде подржан, а није, произвео звук и тиме одао утисак да све ради. Игнорисање је видљиво у логовима и у бројачима, па се таква грешка примећује.

Ово је облик обрасца безбедног стања, дакле стања које се у случају отказа може препознати као безбедно и без ризика [2]. Важно је приметити да безбедно стање није по правилу оно у ком систем ништа не ради, него оно чије су последице познате. Понекад је то мировање, а понекад пуна реакција. Исти образац у другом делу система може носити потпуно супротну одлуку. Када сервис за диоду не може да очита затечено осветљење, он бира пуну јачину, а не пригушену, јер би пригушена диода могла да остане непримећена, а то је овде мање безбедан исход. Слој за логовање понаша се на трећи начин. При отказу прелази на стандардни излаз и наставља, јер би прекид основне функције због помоћне био најгори исход.

Заједничко за сва три случаја није конкретан избор него правило по коме се бира. Када уређај не може да утврди у ком је стању, бира понашање чије су последице познате и најмање штетне, а не оно које изгледа корисно.

4.4 Бројачи стања и извештавање

Оцене из одељка 4.2 немају вредност ако нестају у тренутку настанка. Централни сервис зато води бројаче: укупан број примљених догађаја и по један бројач за сваку од три оцене, затим број покушаних и број неуспелих акција, и најзад име последњег догађаја и његову оцену. Метода `status` над објектом `emergency.notify` те вредности враћа на упит. Бројачи се чувају у радној меморији процеса, па се при његовом поновном покретању враћају на нулу. То је очекивано понашање бројача ове врсте, а не недостатак, јер они описују рад текуће инстанце сервиса, а не историју уређаја. Трајан траг о догађајима чувају логови, који се пишу на партицију која преживљава подизање, о чему је реч у одељку 3.6.

Бројачи акција су намерно збирни, а не раздвојени по сервису. Првобитно решење имало је поља везана за сирену по имену. Са регистром акција то више нема смисла. Раздвојени бројачи значили би да централни сервис поново мора да зна која акција постоји, чиме би се вратила управо она спрега коју регистар уклања. То има своју цену. Из збирног броја неуспелих акција не види се која је акција отказала. Та информација постоји у логовима, где се уз сваки отказ бележи име акције.

Овако скупљене вредности имају двоструку употребу. При раду на уређају оне су најбржи начин да се види да ли сервис уопште добија догађаје и шта са

њима ради. При тестирању оне су мерљив исход. Аутоматизовани тест шаље унапред познат низ догађаја и проверава да ли су се бројачи померили тачно онолико колико је очекивано, што је описано у глави 7.

Глава 5

Имплементација

Одлуке образложене у претходним главама овде добијају свој облик у коду. Није приказан цео код, него само они делови на којима се види како је граница између компоненти спроведена. Имплементација је отвореног кода, објављена под лиценцом MIT (енг. *Massachusetts Institute of Technology License*), и заједно са рецептом (енг. *recipe*) за изградњу, системским јединицама и алатом за тестирање доступна је на сервису GitHub на адреси `https://github.com/dimitrijemarkovic/emergency-notifyd`.

5.1 Структура пројекта и систем изградње

Цео систем гради се као једна целина алатом *CMake*, а унутар ње постоји седам делова, од којих су две библиотеке и пет извршних програма. Сервиси су писани у језику *C*.

Превођењем управљају две опције. Опција `ENABLE_UBUS` укључује ослањање на механизам за међупроцесну комуникацију, а `ENABLE_ZLOG` бира имплементацију слоја за логовање. При превођењу за циљани уређај обе се укључују. Када нису укључене, сервиси се преводе без тих зависности, у облику у ком не раде свој посао, него само омогућавају превођење на рачунару за развој, где магистрала и библиотека за логовање не морају бити доступне.

5.2 Централни сервис

Сервис `emergency-notifyd` при покретању предузима три активности: подиже слој за логовање, поставља бројаче на нулу и повезује се на магистралу.

Затим региструје свој објекат и пријављује се на догађаје, што је приказано на листингу 5.1.

```
1 ret = ubus_add_object(ctx, &notify_object);
2 ...
3 ret = ubus_register_event_handler(ctx, &alarm_event_handler, "alarm
      .*");
```

Листинг 5.1: Регистрација објекта и пријава на конкретну фамилију догађаја

Друга наредба у листингу је место на коме се види природа система. Сервис се не пријављује на појединачне типове аларма него на целу фамилију, обрасцем `alarm.*`. Додавање новог типа аларма зато не тражи никакву измену у пријави на догађаје, јер нов тип стиже до истог руковаоца, а одлука о томе да ли је подржан доноси се тек ниже, у акцијама.

Руковалац догађаја (енг. *event handler*) садржи свега неколико десетина линија, а редослед корака у њему прати главу 4 и одељак 3.3. Најпре се позива провера структуре. Ако она не прође, догађај се означава као неисправан, а разлог одбијања уписује у логове. Ако провера прође, пролази се кроз табелу регистрованих акција, свака се позива са истим аргументима, и памти се да ли је бар једна препознала тип. Тек на крају се бројачи ажурирају и исход уписује у логове.

Објекат `emergency.notify` нуди само једну методу, упит о стању. Њен одговор садржи назив и верзију сервиса, свих шест бројача описаних у одељку 4.4 и податке о последњем обрађеном догађају. Ту се, дакле, види целокупно променљиво стање сервиса, што га чини најбржим начином да се на уређају утврди шта се дешава.

Сервис је једнонитан. Обрада догађаја и одговор на упит одвијају се редом у истој петљи догађаја, а сигнали за прекид рада само заустављају ту петљу, чиме се излаз одвија уредно, кроз ослобађање везе са магистралом и затварање логова.

5.3 Реактивни сервиси

Сервис за сирену и сервис за сигналну диоду имају исту унутрашњу поделу на три дела, описану у одељку 6.1. Средњи део, управљач, зна како се који образац оглашава, дакле којим редом се излаз укључује и искључује и колико свака од тих смена траје. Његов интерфејс приказан је на листингу 5.2.

```
1 int siren_controller_start_by_id(int siren_id, int duration_sec);
2 int siren_controller_stop(void);
3
4 int siren_controller_is_valid_id(int siren_id);
5 const char *siren_controller_pattern_name_by_id(int siren_id);
6
7 int siren_controller_is_running(void);
8 int siren_controller_current_id(void);
9 const char *siren_controller_current_pattern_name(void);
```

Листинг 5.2: Јавни интерфејс дела који управља обрасцима оглашавања сирене

Интерфејс се дели на три групе. Прве две функције мењају стање, наредне две одговарају на питања о исправности задатог обележја, а последње три описују тренутно стање. Управљач за сигналну диоду има интерфејс истог облика, са истим именима осим замене речи која означава уређај. То понављање је намерно, јер управо оно омогућава регистру акција да оба сервиса третира истоветно.

Оглашавање сирене није тренутан догађај него смењивање тонова и тишине које се понавља све док аларм траје, па се извршава у засебној нити. Дељено стање, дакле који образац је у току, колико још треба да се оглашава и да ли се уопште оглашава, заштићено је механизмом узајамног искључивања. Чекање унутар обрасца није једно дугачко успављивање него низ кратких, између којих се проверава да ли образац још треба да траје. Захваљујући томе нова наредба не чека да се претходни образац заврши до краја, него сирена прелази на нов образац релативно тренутно. То понашање потврђено је тестирањем преклапања из одељка 7.3.

Најнижи део је хардверски слој. Оперативни систем приступа сваком излазу кроз посебну датотеку, па се укључивање сирене своди на упис вредности у ту датотеку. Тај слој преводи појмове уређаја у такве уписе. За сирену су то укључивање и искључивање, за диоду постављање јачине појединачног канала. Ниједна путања, број излаза ни вредност времена трајања сигнала не излази изван тог слоја. Све остало у сервису ради са појмовима уређаја, не са његовим регистрима.

5.4 Сервис за објављивање обавештења

Трећи реактивни сервис прати исту трочлану поделу као претходна два, али сваком делу даје другачије значење. Први део нуди `ubus` интерфејс са истим скупом метода који имају и остали. Средњи део, објављивач, држи ред порука и радну нит. Најнижи део, `mqtt_client.c`, скрива клијентску библиотеку протокола MQTT, преко које се поруке шаљу, што је, као што је речено у одељку 3.2, иста улога коју код сирене има слој над `sysfs` путањама.

Разлог за радну нит није исти као код сирене. Код сирене нит постоји зато што образац оглашавања траје. Овде постоји зато што слање поруке може да буде споро или да уопште не успе, јер посредник може бити недоступан, мрежа спора, а покушај повезивања може трајати. Када би се објава обављала у оквиру позива методе, обрада алармног догађаја у централном сервису чекала би мрежу. Уместо тога, метода само уписује запис у ред и одмах се враћа, док објављивање тече у позадини.

Ред је ограничене величине. Када се напуни, најстарији запис се одбацује и броји као неуспео. Тај избор није случајан, јер је у алармном систему новија порука вреднија од старије, а неограничен ред би при дуготрајном отказу мреже растао док не исцрпи меморију уређаја. Одбацивање је, дакле, свесно ограничење са познатим понашањем, а не пропуст.

Реч је о статичком низу од шеснаест места, што за величину једног записа значи мање од пет килобајта заузећа. Пошто сваки аларм производи две поруке, једну о настанку и једну о престанку стања, ред прима осам целих аларма пре него што почне да одбацује најстарије. Већи ред не би нужно био бољи, јер би после повратка мреже највероватније испоручио и обавештења о догађајима који су одавно прошли, што претплатнику не говори о тренутном стању уређаја.

Одредиште порука није уграђено у код. Адреса посредника се чита из окружења при покретању сервиса, а системска јединица је узима из засебне датотеке са подешавањима, па сервис не зна ништа о томе где се посредник налази ни коме он поруке даље прослеђује. Усмеравање обавештења ка примаоцу изван уређаја, било да је реч о електронској пошти, SMS порукама или обавештењима у мобилној апликацији, зато тражи измену подешавања, а не измену кода. То је и проверено на уређају, тако што је сервису задата адреса посредника на другом рачунару у мрежи, а поруке су затим примљене на том

рачунару.

Свака порука објављује се на теми изведеној из типа аларма, а њен садржај носи податке о догађају и ознаку стања. Престанак алармног стања, дакле тренутак када опасности више нема, не преноси се изостанком поруке него новом поруком која јавља да је стање отклоњено. Разлог томе је да претплатник који је примио обавештење о аларму мора сазнати и да је аларм престао, јер га супротно оставља у уверењу да опасност траје.

Интерфејс слоја за логовање има четири нивоа логова и једну функцију за покретање, којој се прослеђује име сервиса. То име постаје категорија под којом се логови разврставају. Тај интерфејс има две имплементације. Једна уписује на стандардни излаз и увек је доступна, друга користи библиотеку *zlog* и преводи се само када је та опција укључена.

Разврставање по сервису изведено је у подешавању библиотеке, приказаном на листингу 5.3.

```
1 notifyd.*  "путања<>/notifyd.log", 1MB * 5 ~ "notifyd.log.#r"
2 siren.*    "путања<>/siren.log",   1MB * 5 ~ "siren.log.#r"
3 led.*      "путања<>/led.log",     1MB * 5 ~ "led.log.#r"
4 mqtt.*     "путања<>/mqtt.log",    1MB * 5 ~ "mqtt.log.#r"
```

Листинг 5.3: Правила за разврставање логова по сервису

Свако правило повезује категорију са сопственом датотеком и одређује ротацију, тако да се датотека преименује када пређе један мегабајт, а чува се пет старијих. Сервиси тако не пишу у исту датотеку, па се логови једног сервиса могу читати без филтрирања туђих порука. Одредиште је на партицији која преживљава подизање уређаја, а системске јединице чекају да се она монтира пре него што сервиси крену.

5.5 Паковање и покретање на уређају

Паковање (енг. *packaging*) и испорука на уређај изведени су рецептом за систем изградње *Yocto*. Рецепт преузима изворни код непосредно из складишта, преводи га са обе опције укључене и уграђује у слику пет извршних програма, четири системске јединице и подешавања за логовање и за адресу посредника. Зависности од магистрале, помоћних библиотека и библиотеке за логовање наведене су и за изградњу и за рад, чиме их систем изградње сам повлачи у слику.

Пакет је остао један, иако гради четири сервиса. Разлагање на засебне пакете по извршном програму било би изводљиво, али би додало сложеност без неке користи, јер прототип увек испоручује сва четири заједно и не постоји случај у ком би један ишао без осталих.

Уз сваки сервис иде и системска јединица, датотека у којој пише како се тај сервис покреће и под којим условима. Свака описује сервис као обичан процес који се поново подиже ако се неочекивано заустави, са кратким размаком између покушаја. У тим датотекама записано је и то да је пожељно да централни сервис крене после посредника порука и после сва три реактивна сервиса, али то није услов. Ако неки од њих закасни, централни сервис свеједно креће. Разлог је што он објекте реактивних сервиса тражи тек када обрађује догађај, а не када се подиже.

Провером на уређају потврђено је да јединица која покреће посредника порука постоји под тим именом, ради и подиже се сама, па све три јединице заиста показују на постојећу јединицу.

Самостално подизање је потврђено тестирањем. После поновног покретања уређаја сви сервиси ушли су у активно стање непосредно по подизању система, без иједног ручног покретања, а њихови објекти били су одмах видљиви на магистрали. Тврдња да је систем применљив тиме не почива на томе како је замишљен, него на томе како се показао на уређају.

Од места на ком се системске јединице налазе у систему датотека зависи да ли ће их систем за управљање сервисима уопште видети. Пошто је корени систем датотека, како је већ речено, непроменљив, директоријум са подешавањима, уобичајено `/etc`, налази се у засебном слоју који се прикључује изнад њега, и то тек у току покретања уређаја. Систем за управљање сервисима до тог тренутка већ прочита које јединице постоје и које су означене као тражене, па оне које се налазе у том слоју за њега још не постоје. Тестирање је показало да зато није довољно ни да се јединица упише и уобичајеном командом означи као тражена, јер и та ознака завршава у истом слоју. Јединице морају бити уграђене у саму слику система, у део стабла доступан од првог тренутка покретања, што рецепт и ради, а ручно пребацивање датотека при развоју не.

Глава 6

Проширивост система

Систему су после првог реактивног сервиса додата још два. Ова глава описује поступак по коме се додају, шта је свако од та два додавања захтевало и шта се при томе показало. По истом поступку додао би се и сваки следећи сервис, на пример онај који укључује осветљење у стану, искључава излазе ради евакуације или покреће одвођење дима.

6.1 Образац за додавање новог реактивног сервиса

Нов реактивни сервис уводи се у четири корака, а ниједан од њих не мења постојећу обраду догађаја.

- **Сам процес** - сервис добија сопствени извршни програм и сопствену унутрашњу поделу на три дела, исту ону коју већ има сирена, дакле датотеку која излаже `ubus` интерфејс, управљач који води логику и хардверски слој који једини познаје путање уређаја. Ова подела није формално правило система него уочен образац.
- **Објекат на магистрали** - сервис региструје `ubus` објекат сопственог имена и на њему исти скуп метода који постоји код осталих реактивних сервиса, дакле покретање реакције, њено отказивање и упит о стању. Само тако централни сервис може све акције да обрађује истим кодом.
- **Акција у централном сервису** - танак слој који преводи тип аларма у позив ка новом објекту и враћа једну од три вредности типа `action_`

`result_t`. Ту стоји и списак типова аларма које тај сервис препознаје, у складу са поделом описаном у одељку 4.2.

- **Ред у табели `alarm_actions`** - тиме је сервис укључен у обраду. Датотека `notifyd_ubus.c`, у којој се догађаји обрађују, не мења се уопште.

Табела акција је место на коме се бира коме ће командна порука, о којој је реч у одељку 3.4, бити послата.

Образац има и границу. Регистар је статичка табела која се чита у време превођења, па додавање сервиса и даље значи ново превођење централног сервиса. Учитавање акција у време извршавања било би могуће, али би тражило механизам за динамичко учитавање кода и проверу његове исправности, што за уређај ове намене доноси више ризика него користи.

6.2 Сервис за сигналну диоду као провера обрасца

Сервис `emergency-led-service` додат је уз четири новине и ниједну измену затеченог тока обраде. Настао је нов процес са истом унутрашњом поделом на три дела, где `led_service.c` излаже `ubus` интерфејс, `led_controller.c` води логику, а `led_hw.c` једини познаје путање уређаја. Регистрован је објекат `emergency.led` са истим скупом метода који има и сирена. У централном сервису додата је датотека `led_action.c` са преводом типа аларма у позив ка том објекту. И, најзад, у табелу `alarm_actions` додат је један ред.

Датотека `notifyd_ubus.c`, у којој се алармни догађаји примају и обрађују, није измењена. Обрада догађаја не помиње ниједну појединачну акцију ни пре ни после додавања другог сервиса.

Уз то додавање ишле су и две одлуке о опсегу. Прва је да сервис дира само статусну диоду, а не и диоду напајања, зато што је реч о реакцији на аларм. Друга је да `emergency.led` преслика интерфејс сирене уместо да понуди сопствени облик у ком се стање поставља једним позивом. Управо то што сви нуде исти интерфејс омогућава регистру да све акције третира истоветно.

6.3 Сервис друге врсте као стварна провера

Сигнална диода је, међутим, слаба провера обрасца, јер је структурно иста са сиреном, дакле локални хардвер, тренутан ефекат и реакција одређена само типом аларма. Права провера тражи сервис који ради нешто друго.

Трећи реактивни сервис је прва акција друге природе. Он не делује на хардвер него шаље податак са уређаја, његов ефекат није тренутан, и може отказивати и опорављати се. Управо зато што се не уклапа савршено у постојећи образац, његово додавање је открило три налаза које се на прва два сервиса нису могли видети.

Интерфејс акција био је преузак. Сирени и диоди довољан је тип аларма, јер из њега следи који образац треба покренути. Сервису за обавештавање треба *садржај* догађаја: шта се догодило, колико је озбиљно, одакле долази. Ти подаци су постојали од самог почетка, јер их централни сервис издваја и проверава пре него што ишта предузме, али их до тада ниједна акција није користила. Потпис акције морао је бити проширен да их прослеђује. Интерфејс је дотле преносио мање него што је могао, а то се није видело код друга два реактивна сервиса.

Престанак алармног стања носи информацију. Код сирене и диоде престанак значи престанак опасности. Код обавештења порука је већ отишла и не може се повући. Престанак зато не значи да се ништа не предузима, него да се шаље нова порука која јавља да је опасност прошла. Заједнички интерфејс ту не одговара сасвим, јер иста метода има битно различито значење за различите врсте акција. Та операција зато значи „заврши текуће стање на начин који том одредишту има смисла”, а не „заустави излаз”.

Значење бројача се променило. Пошто позив ка сервису за обавештавање не чека да порука буде испоручена, бројач послатих акција у централном сервису за ту акцију броји предато на обраду, а не испоручено. За сирену и диоду синхрони позив јесте потврда извршења, а за мрежну акцију није. Податак о томе да ли је порука испоручена налази се у стању самог сервиса за обавештавање, а не у бројачима централног сервиса. Исти бројач тако за две акције значи једно, а за трећу друго.

Прва два налаза су и исправљена. Потпис акције проширен је тако да поред типа аларма прослеђује и садржај догађаја, чиме је интерфејс почео да преноси онолико колико је од почетка имао, а отказивање код сервиса за

обавештавање објављује поруку о отклањању уместо да ћути. Трећи налаз није исправљен него признат и наведен на месту где се бројачи објашњавају, јер се разлика између предатог и испорученог не може уклонити тако што ће се сакрити. Ниједан од ова три налаза не би се појавио да је трећи сервис био још један управљач локалним хардвером.

Глава 7

Тестирање и евалуација

Алармни догађај не враћа одговор пошиљаоцу, па се исправност обраде проверава преко стања реактивних сервиса и преко бројача централног сервиса. Ова глава наводи шта је испитано, под којим условима, и који су резултати добијени на циљној плочи.

7.1 Аутоматизовани симулатор

Алат `emergency-notify-sim` је обичан `ubus` клијент. Он не региструје сопствени објекат, па га ниједан сервис не види као учесника у систему и његово присуство не мења оно што се мери.

Свих седамнаест тест случајева изведено је овим алатом, аутоматски. Једини изузетак је провера изолације отказа, која тражи гашење и покретање системских сервиса, па је изведена ручно.

Тестирање најпре проверава редован рад, дакле да исправан аларм покрене баш ону реакцију која му је додељена, а тек затим одступања од тога. Сваки тест има три корака. Систем се прво доводи у познато стање, тако што се на реактивним сервисима прекине реакција која евентуално још траје. Затим се шаље алармни догађај, исто онако како би га послао прави извор аларма. На крају се, пошто догађај не враћа одговор, преко метода `status` чита стање реактивних сервиса и упоређује са обрасцем који је за тај тип аларма предвиђен.

Бројачи централног сервиса кумулативни су од покретања процеса, па симулатор пореди њихов *прираст*, а не апсолутне вредности. Тест је зато тачан без обзира на то колико дуго сервис већ ради.

Резултат сваког теста исписује се на стандардни излаз, уз завршни збир прошлих и палих тестова.

7.2 Методологија тестирања

Скуп тест случајева покрива три групе улаза, које одговарају трима оценама из одељка 4.2. Прву чине пет исправних аларма (пожар, провала, паника, излив воде и медицински позив) за које се проверава да је покренут тачно онај образац који им је додељен. Другу чине три догађаја која морају бити игнорисана, и то догађај облика `alarm.*` чији тип није подржан, догађај потпуно непознатог имена, и догађај који уопште не припада фамилији `alarm.*`. Трећу чине четири неисправна садржаја, и то изостављено обавезно поље, недозвољена вредност озбиљности, неслагање поља `type` са именом догађаја и потпуно празан садржај.

Изостанак реакције на непознат аларм је пројектна одлука, а не пропуст, па се друга група проверава једнако пажљиво као и прва.

Уз њих иду три теста која проверавају понашање, а не појединачни улаз.

Прво је преклапање аларма. Шаље се `alarm.fire`, а одмах затим `alarm.panic`, без отказивања између њих, и проверава се који је образац на крају затечен. Очекује се образац другог аларма, јер реактивни сервис пре покретања новог обрасца отказује текући.

Правило да последњи догађај преузима уређај свесна је одлука која је донета ради предвидљивости, јер у сваком тренутку постоји тачно један образац који се оглашава и он одговара последњем аларму. Донета је и на основу стварних корисничких захтева уочених у индустријској пракси. Уређивање аларма по озбиљности било би могуће, пошто поље `severity` већ постоји.

Друго и треће су непосредни неисправни позиви ка реактивним сервисима, мимо централног сервиса. Симулатор позива методу `alarm` над објектима `emergency.siren` и `emergency.led`, прво са непостојећим идентификатором, а затим са празним садржајем. Сервис мора да врати грешку и да после тога и даље одговара на упит о стању. Реактивни сервис свој интерфејс излаже свакоме на магистрали, па не сме да рачуна на то да улаз долази проверен.

Провера изолације отказа, дакле гашење једног реактивног сервиса па слање аларма, изведена је ручно на плочи и није део симулатора.

7.3 Резултати на циљној плочи

Тестирање је изведено на циљној плочи и више пута поновљено. Сваки пут су прошли сви тест случајеви, њих седамнаест, без иједног неуспеха. Сва четири сервиса су се подигла, а сва четири `ubus` објекта била су регистрована и видљива на магистрали.

Стање бројача после прелаза дато је у табели 7.1. Збир прихваћених, игнорисаних и неисправних догађаја поклапа се са укупним бројем примљених догађаја.

Табела 7.1: Стање бројача централног сервиса после прелаза

Бројач	Вредност
примљени догађаји	15
прихваћени	9
игнорисани	2
неисправни	4
послате акције	27
неуспеле акције	0

Седамнаест тест случајева дало је петнаест догађаја. Првих дванаест тестова шаљу дванаест догађаја, али један од њих није из фамилије `alarm.*` и зато уопште не стиже до руковаоца у централном сервису, па остаје једанаест. Тест преклапања шаље још два, а тест отказа посредника порука још два. Два теста су непосредни позиви метода, без иједног догађаја, а последњи тест само чита бројаче.

Девет прихваћених догађаја пута три регистроване акције даје двадесет седам позива, и управо толико их је забележено. Регистар описан у одељку 3.3 тиме сваки прихваћени догађај заиста распоређује на сва три реактивна сервиса, што проширивост чини провереном особином система у раду, а не тврдњом о изворном коду.

Пресликавање типа аларма на обрасце потврђено је на уређају, за све реактивне сервисе истовремено, и дато је у табели 7.2. Имена образаца су ознаке под којима они стоје у коду и описују сам ритам, па је `temporal_3` ритам од три кратка тона пред паузу, поменут у уводу, `panic_pulse` брзо непрекидно пулсирање, а `red_fast` брзо треперење црвеном бојом. Нема ниједног типа код кога је реаговао само део њих.

Табела 7.2: Пресликавање типа аларма на обрасце, потврђено на хардверу

Догађај	Сирена	Сигнална диода
`alarm.fire`	`temporal_3`	`red_fast`
`alarm.burglary`	`panic_pulse`	`red_solid`
`alarm.panic`	`panic_pulse`	`red_solid`
`alarm.water_leak`	`temporal_4`	`blue_fast`
`alarm.medical`	`medical_slow_pulse`	`blue_slow`

Улази који не смеју ништа да покрену заиста нису ништа покренули. Неподржани типови аларма, догађај ван фамилије `alarm.*` и сва четири неисправна садржаја оставили су сва три реактивна сервиса без покренуте реакције. У логовима су се појавила сва четири разлога одбијања која постоје у коду (недостатак обавезног поља, недозвољена озбиљност, неслагање типа са именом догађаја и потпуно празан садржај). Дакле, свака од четири гране одбијања јавила се и у стварном раду, а не само у коду.

Тест преклапања потврдио је политику описану у одељку 7.2. После слања првог аларма сирена је свирала образац додељен пожару, а после другог аларма, послатог без икаквог отказивања између, затечен је образац додељен паници. Последњи догађај је преузео уређај, како је и предвиђено.

Оба реактивна сервиса одбила су непосредне неисправне позиве (и оне са непостојећим идентификатором и оне са празним садржајем), вративши грешку са разлогом, и после тога су и даље одговарала на упит о стању. Отпорност на неисправан позив који не долази кроз централни сервис тиме је потврђена за оба сервиса.

Обавештења послата трећим реактивним сервисом проверена су на одредишту, а не преко бројача. Осматрањем саобраћаја на посреднику ухваћене су стварне поруке, приказане на листингу 7.1.

```
1 emergency/alarm/fire
2   {"state":"active","type":"fire","severity":"critical",
3    "source":"emergency-notify-sim","message":"Simulated fire alarm"}
4 emergency/alarm/fire
5   {"state":"cleared","type":"fire"}
6 emergency/alarm/burglary
7   {"state":"active","type":"burglary","severity":"high"}
8 emergency/alarm/water_leak
9   {"state":"active","type":"water_leak"}
```

Листинг 7.1: Поруке ухваћене на посреднику током тестирања

Свака врста аларма објављује се на својој адреси на посреднику, а уз сваки аларм иде и порука о његовом отклањању. Тиме је потврђено да се и престанак аларма објављује као податак, уместо да прође нечујно, како је описано у одељку 5.4. Приказано је, дакле, да порука постоји на магистрали, а не само да је бројач порастао.

Најзад, настале су четири одвојене датотеке са логовима, по једна за сваки сервис, како је и подешено.

7.4 Изолација отказа

Изолација отказа проверена је гашењем појединачних компоненти. Тестирање је изведено ручно на уређају, из разлога наведеног у одељку 7.2, у четири корака.

У првом кораку заустављен је сервис сирене, па је послат исправан алармни догађај. Догађај је обрађен и означен као прихваћен, две акције су успеле а једна отказала, што су бројачи и показали. Број прихваћених порастао је за један, број послатих акција за два, а број неуспелих акција за један. Сигнална диода се огласила обрасцем предвиђеним за пожар, обавештење је објављено, а централни сервис остао је у раду. Цео ток види се у логовима, приказаним на листингу 7.2.

```
1 siren action resolved: event=alarm.fire, siren_id=2, duration=10,
    pattern=temporal_3
2 ERROR siren action failed: object 'emergency.siren' not found: Not
    found
3 led action resolved: event=alarm.fire, led_id=1, duration=10,
    pattern=red_fast
4 led alarm request sent: led_id=1, duration=10, pattern=red_fast
5 mqtt alarm request sent: event=alarm.fire
6 event classified: event=alarm.fire result=accepted
```

Листинг 7.2: Логови централног сервиса при отказу једног реактивног сервиса

Ових неколико редова листинга садржи целу тврдњу овог рада о изолацији. Прва акција је покушала и пријавила да тражени објекат не постоји. Преостале две наставиле су као да се ништа није догодило и извршиле своје

реакције. Догађај је на крају обрађен до краја и оцењен као прихваћен. Отказ једног реактивног сервиса није зауставио ни обраду ни преостале реакције.

Овде је отказао сервис, дакле софтверски део. Отказ на страни хардвера види се на исти начин, зато што акција враћа исти исход без обзира на то да ли је сервис недоступан или упис на излаз није успео. Такав случај је и забележен на уређају и описан је у одељку 7.5.

У другом кораку сервис сирене је поново покренут. Наредни догађај произвео је три успешне акције. Веза ка реактивном сервису тражи се, дакле, при свакој обради догађаја, а не једном при покретању, па опоравак не захтева поновно покретање централног сервиса.

Трећи корак поновио је тестирање у другом смеру. Угашен је сервис `emergency-led-service`, а сирена се огласила обрасцем предвиђеним за медицински позив, док је број неуспелих акција порастао за један. Изолација, дакле, не зависи од тога која је акција отказала ни од њеног места у табели.

Посебно је проверен и отказ посредника порука, зато што је он друге природе од отказа сервиса. Сервис ради, али одредиште коме шаље није доступно. Сирена и диода реаговале су нормално, обрада догађаја текла је до краја, а сервис за обавештавање забележио је неуспех и остао у раду. По враћању посредника, наредна обавештења су објављена без поновног покретања иједног сервиса.

Четврти корак је гранични случај. Угашена су сва три реактивна сервиса, па је послат исправан алармни догађај. Догађај је означен као прихваћен, ниједна акција није успела, а број неуспелих акција порастао је за три. Систем је, дакле, забележио прихваћен догађај иако се физички није догодило ништа.

То није недоследност него последица поделе описане у одељку 4.2. Оцена описује догађај, а он јесте био исправан и јесте био препознат. Успешност извршења прати се одвојено, бројачем неуспелих акција. Мешањем то двоје изгубила би се разлика између неисправног улаза и исправног улаза на који хардвер није одговорио. Цена таквог избора је што оцена „прихваћен” не потврђује да је реакција стварно изведена на хардверу, па је при тумачењу стања треба читати заједно са бројем неуспелих акција. Ово понашање предвиђено је при пројектовању, а овим тестирањем је и потврђено.

Уз изолацију су проверене и две особине система.

- **Опоравак после пада процеса** - сваки сервис је насилно прекинут, чиме је заобиђено свако уредно гашење. Сви су враћени у рад са новим

идентификатором процеса.

- **Поштовање задатог трајања** - аларму је задато трајање од десет секунди. Непосредно после слања и сирена и сигнална диода биле су у раду, а по истеку задатог трајања обе су се саме зауставиле, без спољне наредбе.

7.5 Ограничења и налази

Ниједан тест није пао, али је тестирање изнело четири налаза који мењају оно што се на основу њега сме тврдити. Два су у међувремену исправљена и провера је поновљена, док два остају ограничења система.

Прва је била нетачна претпоставка о опсегу вредности сигналне диоде. Код за пун интензитет уписивао је вредност већу од дозвољене, а систем је радио исправно само зато што управљачки програм одсеца вредност на дозвољени максимум, дакле тачно, али из погрешног разлога. Хардверски слој сада при покретању очитава највећу дозвољену вредност са самог уређаја и из ње изводи и пуну и пригушену јачину. Провером на плочи очитана вредност поклапа се са оном коју уређај пријављује, па претпоставке више нема.

Друга се тиче облика записа у дневнику. Подешени формат није примењен, па логови носе уграђени подразумевани облик библиотеке, у ком се име категорије не види унутар самог записа. Раздвајање логова по сервису тиме није угрожено, јер сваки сервис пише у сопствену датотеку, али подешавање формата захтева проверу. Уз њега иде и последица саме апстракције, објашњена у одељку 3.6. Подаци о месту позива увек показују на омотач.

Трећа је најозбиљнија за тумачење резултата. Сирена се укључује преко излаза на плочи, дакле једне управљачке линије и једног излаза са променљивим односом трајања импулса, којима хардверски слој приступа кроз систем датотека језгра оперативног система. Ти излази затечени су као већ заузети, па их сервис није могао преузети. Иницијализација то стање пријављује, али рад наставља. Провером на уређају утврђен је и узрок. На слици оперативног система већ раде сервиси произвођача платформе који управљају истом сиреном и истом сигналном диодом, а међу изведеним излазима налази се тачно онај који користи хардверски слој нашег сервиса сирене. Прототип и затечени сервиси, дакле, истовремено полажу право на исти излаз, а шта хардвер ради када му два програма приступају упоредо није испитано. Саме

реакције ипак нису остале непроверене. На уређају са прикљученом сиреном непосредно је опажено да се она заиста оглашава при алармном догађају и да сигнална диода при томе светли. Оно што остаје отворено није, дакле, да ли реакција постоји, него како се два власника истог излаза понашају када оба покушају да га користе.

Четврта је била трајност логова. Прва провера после поновног покретања (енг. *reboot*) уређаја показала је да дневници не преживљавају подизање, јер је путања којом су писани била у радној меморији. Провером на плочи утврђено је да је корен система датотека монтиран само за читање и да `/var/log` води у привремени систем датотека, па је одредиште премештено на партицију која преживљава подизање. Поновљена провера после поновног покретања затекла је записе настале пре њега.

Глава 8

Закључак

У оквиру овог мастер рада предложена је, образложена и на стварном хардверу проверена архитектура модуларног система за обраду и сигнализацију алармних догађаја у окружењу са уграђеним рачунаром. Поред саме архитектуре, у раду су изложени и принципи на којима она почива, а то су модуларна декомпозиција по критеријуму сакривања информација, обрада вођена догађајима, валидација улазних догађаја и тестирање. Ти принципи су познати, али је њихова примена овде посматрана у контексту безбедносних уређаја за паметне куће, где погрешна реакција на неразумљив улаз није само програмска грешка него безбедносни проблем.

Допринос рада је архитектура алармног система која је истовремено лако проширива и предвидљива при отказу, а прилагођена ограничењима уграђеног Линукс окружења. Одлука о томе шта треба предузети раздвојена је од начина на који се то изводи, тако да централни сервис не познаје унутрашњи рад реактивних сервиса, нити они познају правила по којима се одлучује.

У раду су образложена и аутоматизованим испитивањем на развојној плочи проверена четири својства система. Безбедност се огледа у томе што непознат или неисправан улаз не покреће ниједну реакцију, јер се сваки догађај проверава пре него што ишта буде предузето. Поузданост је потврђена гашењем појединачних реактивних сервиса на уређају, при чему обрада догађаја није прекинута, а преостали сервиси су наставили да раде. Примењивост је показана радом на стварном уређају, где се систем испоручује као пакет са системским јединицама и подиже се сам после поновног покретања, без иједне ручне радње. Проширивост је остварена регистром акција, у који се нова реакција уводи једним уписом, док њено понашање остаје у засебном модулу,

тако да део који прима и обрађује догађаје остаје неизмењен, а потврђена је накнадним додавањем нових реактивних сервиса.

Систем се даље може развијати у неколико праваца. Први је увођење реактивних сервиса који управљају извршним органима изван самог уређаја, попут електромагнетних брава. Постојећа три сервиса делују на локални хардвер или шаљу обавештење, док би овакав сервис издавао команду другом уређају, чиме би се показало да предложена архитектура може да управља и периферним уређајима у систему паметне куће. Други је рад у стварном окружењу, где догађаје не производи симулатор него прикључени сензори и детектори, што би показало и како се систем понаша при улазу који не долази из унапред припремљеног низа. Трећи је проширење скупа препознатих аларма. Систем тренутно реагује на пет врста, што је довољно да се архитектура провери, али не и за уређај који би радио у стварном систему паметне куће, где се врсте опасности разликују од произвођача до произвођача и временом се допуњују.

Литература

[1] Mouiad Al-Wahah and Auhood Al-Hossenat. Safety Assurance in IoT-Based Smart Homes. In Yu Chen and Ronghua Xu, editors, *Edge Computing Architecture — Architecture and Applications for Smart Cities*. IntechOpen, 2024. DOI: 10.5772/intechopen.1005492.

[2] Ashraf Armoush. *Design Patterns for Safety-Critical Embedded Systems*. PhD thesis, RWTH Aachen University, 2010. on-line at: `https://d-nb.info/1007034963/34`.

[3] F. Bachmann, L. Bass, and R. Nord. Modifiability Tactics. Technical Report CMU/SEI-2007-TR-002, Software Engineering Institute, Carnegie Mellon University, 2007. on-line at: `https://www.sei.cmu.edu/library/modifiability-tactics/`.

[4] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock. Quality Attributes. Technical Report CMU/SEI-95-TR-021 (ESC-TR-95-021), Software Engineering Institute, Carnegie Mellon University, December 1995. on-line at: `https://www.sei.cmu.edu/documents/1142/1995_005_001_16427.pdf`.

[5] P. Th. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[6] European Committee for Electrotechnical Standardization (CENELEC). EN 50131-1:2006+A3:2020 — Alarm systems — Intrusion and hold-up systems — Part 1: System requirements, 2020. серија стандарда у више делова; наведен је први део, са изменама A1:2009, A2:2017 и A3:2020.

[7] Gregor Hohpe and Bobby Woolf. Enterprise Integration Patterns — Messaging Patterns Catalogue, 2025. on-line at: `https://www.enterpriseintegrationpatterns.com/patterns/messaging/`.

[8] International Electrotechnical Commission. Overview of IEC 61508 and Functional Safety, 2022. on-line at: `https://assets.iec.ch/public/acos/IEC%2061508%20&%20Functional%20Safety-2022.pdf`.

[9] Tammy Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes, 2005.

[10] OASIS. MQTT Version 5.0. OASIS Standard, 2019. on-line at: `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html`.

[11] OpenWrt Project. ubus (OpenWrt micro bus architecture) — Technical Reference, 2025. on-line at: `https://openwrt.org/docs/techref/ubus`.

[12] D. L. Parnas. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[13] Hardy Simpson. zlog User's Guide, 2025. on-line at: `https://hardysimpson.github.io/zlog/UsersGuide-EN.html`.

[14] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen. The Structure and Value of Modularity in Software Design. In *Proceedings of the 8th European Software Engineering Conference held jointly with the 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, 2001. стр. 99–108.

[15] The Yocto Project. Yocto Project Documentation, 2026. on-line at: `https://docs.yoctoproject.org/`.

[16] A. Wright, H. Andrews, and B. Hutton. JSON Schema Validation: A Vocabulary for Structural Validation of JSON, 2022. Internet-Draft draft--bhutton-json-schema-validation-01; on-line at: `https://json-schema.org/draft/2020-12/json-schema-validation`.

Биографија аутора

Димитрије Марковић рођен је 10. октобра 2000. године у Брусу. Основне студије завршио је 2024. године на Математичком факултету Универзитета у Београду, на смеру Информатика, са просечном оценом 8,67, и тиме стекао звање дипломираног информатичара. Након тога уписао је мастер студије на истом факултету.

Од 2022. до 2023. године био је запослен у компанији Orion Innovation, а од 2025. године ради у компанији Ikotek.