REPUBLIKA E SHQIPËRISË
UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I TEKNOLOGJISË SË INFORMACIONIT
DEPARTAMENTI I INXHINIERISË INFORMATIKE

KUJTIM HYSENI

PËR MARRJEN E GRADËS
"DOKTOR"
NË "TEKNOLOGJITË E INFORMACIONIT DHE KOMUNIKIMIT"
DREJTIMI "INXHINIERI INFORMATIKE"

DISERTACION

IMPLEMENTIMI I ARKITEKTURAVE TË ORIENTUARA NË SHËRBIME NË PAJISJE TË MBYLLURA PËRMES GJENERATORËVE TË KODIT

Udhëheqës shkencor
Akad. Prof. Dr. Neki Frashëri

Tiranë, 2019

IMPLEMENTIMI I ARKITEKTURAVE TË ORIENTUARA NË SHËRBIME NË PAJISJE TË MBYLLURA PËRMES GJENERATORËVE TË KODIT

Disertacion
i paraqitur në Universitetin Politeknik të Tiranës
për marrjen e gradës
"Doktor"
në:
"Teknologjitë e Informacionit dhe Komunikimit"
Drejtimi "Inxhinieri Informatike"

Nga
Z. Kujtim Hyseni
2019

JURIA PËR VLERËSIMIN E DISERTACIONIT PËR FITIMIN E GRADËS SHKENCORE "DOKTOR"

Miratuar

me vendimin e Këshillit të Profesorëve të FTI-së Nr._____, datë_____:

Kryetar i Jurisë_____

Anëtar i Jurisë_____

Anëtar i Jurisë_____

Anëtar i Jurisë_____

Anëtar i Jurisë_____

Dekan i Fakultetit të Teknologjisë së Informacionit

_____

TABELA E PËRMBAJTJES

LISTA E FIGURAVE

LISTA E TABELAVE

LISTA E LISTINGJEVE

KUSHTUAR

Jonit!

FALËNDERIME

Fillimisht, falënderoj udhëheqësin shkencor Akad. Prof. Dr. Neki Frashërin. Pa të ky punim nuk do të mund të realizohej. Përveç aspektit shkencor, ai më ka dhënë edhe guxim dhe përkrahje morale të pazëvendësueshme.

Falënderoj edhe përgjegjësen e Departamentit të Inxhinierisë Informatike, Prof. Asoc. Dr. Elinda Kajo Meçe për këshillat e vlefshme si edhe profesorët tjerë për konstruktivitetin e treguar gjatë ndjekjes së Shkollës së Doktoraturës.

Në fund, falënderoj familjen time për përkrahjen, jo vetëm tani, por që nga fillimi i rrugëtimit tim shkencor.

Kujtim Hyseni

Nëntor 2019, Tiranë, Shqipëri

PËRMBLEDHJE

Ky punim ka të bëjë me integrimin e pajisjeve të mbyllura në sisteme të bazuar në Arkitektura të Orientuara në Shërbime (AOSh). Integrimi nënkupton implementimin e teknologjive të bazuara në AOSh, si CORBA dhe Shërbimet Ueb, në pajisje të mbyllura. Kjo thjesht do të thotë që pajisja e mbyllur të ekzekutojë kodin që do ta bënte atë një servis CORBA apo Shërbim Ueb, që do ti përgjigjej kërkesave të klientit. Megjithëse për pajisje të fuqishme si kompjuterët personal kjo mund të konsiderohet punë e kryer, integrimi i pajisjeve të mbyllura mbetet sfidë. Kjo për arsye se teknologjitë e përmendura të orientuara në shërbime kërkojnë mjaft burime si kujtesë programi dhe RAM, që janë të mangëta te pajisjet e mbyllura. Në këtë mënyrë pamundësohet realizimi i tyre në pajisje të mbyllura siç janë mikrokontrollorët. Roli i mikrokontrollorit është që të shkëmbejë të dhëna me pajisjen e fuqishme si kompjuteri personal p.sh. duke pranuar komanda nga kompjuteri dhe aplikojë sinjalet përkatëse në sensorët apo aktuatorët e lidhur për mikrokontrollor. Ndërtimi i sistemit të bazuar në AOSh mundëson zgjerimin e mëtejshëm të sistemit dhe mirëmbajtje në mënyrë të standardizuar. Në literaturë aplikohet qasja e rëndomtë dhe ajo e komprimuar. Në qasjen e rëndomtë, nuk ekziston kompilatori për përkthimin e kodit të gjeneruar nga veglat përkatëse siç është p.sh. gSOAP në kod të ekzekutueshëm 8051. Te ajo e komprimuar gjenerohet kod i tepërt i manipulimit në nivel bitësh, siç janë zhvendosja e bitëve, krahasimi dhe aplikimi i operacioneve logjike. Në të dyja rastet, kodi rezultues është tepër i madh për tu programuar në kujtesën tepër të vogël prej 12 kilobajtëve të mikrokontrollorit 8051 që e përdorim. Kjo i bën të papërshtatshme që të dyja qasjet dhe ne propozojmë qasje të re me burime dukshëm më të vogla. Qasja e re bazohet në ndarje të përgjegjësive në mes të pajisjes së fuqishme siç është kompjuteri personal dhe mikrokontrollorit.

Ne propozojmë zgjidhjen duke e futur konceptin e serverit lidhës. Serveri lidhës ekzekutohet në pajisje të fuqishme si kompjuteri personal, dhe merret me punë komplekse duke e lënë mikrokontrollorin të merret me ato të thjeshta. Klienti e sheh vetëm serverin lidhës, i cili, kërkesat e nivelit të lartë nga klienti i përkthen në të thjeshta duke ia përcjellur më tej serverit në mikrokontrollor. Klienti dhe serveri

lidhës komunikojnë përmes protokollit specifik për teknologjinë që janë të realizuar. Kështu, për teknologjinë CORBA ata komunikojnë bazuar në protokollin IIOP (ang. Internet Inter-Orb Protocol), ndërsa për Shërbime Ueb komunikojnë përmes SOAP (ang. Simple Object Access Protocol). Në anën tjetër, serveri lidhës dhe ai në pajisje të mbyllur (mikrokontrollor) komunikojnë përmes protokollit që ne e kemi projektuar, të quajtur AMISP (ang. Adaptive Minimal Inter Server Protocol). Pra, komunikimi në mes serverit lidhës dhe atij në pajisje të mbyllur nuk varet nga teknologjitë e implementimit të klientit dhe serverit lidhës. Në këtë mënyrë është bërë edhe e mundur që serverit të mbyllur, të njëjtit, në të njëjtën kohë të iu qasemi nga teknologji të ndryshme. E gjithë kjo punë e bërë manualisht do t'iu ishte ekspozuar gabimeve, kohës së gjatë si dhe njohurive të detajizuara teknike të nevojshme. Për këtë, ne kemi zhvilluar gjeneratorë të kodit që do të gjeneronin kodin e serverit lidhës dhe atë të pajisjes së mbyllur. Serveri lidhës gjenerohet i tëri dhe varet nga teknologjia, p.sh. është tjetër për JacORB (implementim në Java i CORBA-s) dhe tjetër për JAX-WS (implementim në Java i Shërbimeve Ueb). Në të vërtetë, për gjuhën e dhënë programuese ata janë shumë të ngjashëm dhe mund të transformohen prej njërit në tjetrin duke i modifikuar vetëm disa rreshta të kodit burimor. Për serverin në pajisje të mbyllur gjenerohet vetëm komunikimi me serverin lidhës, më të cilin rëndom i shkëmbejnë parametrat e operacioneve. Çka mbetet të shkruhet nga zhvilluesi i sistemit është implementimi i operacioneve për çka ato janë të destinuara në serverin e mbyullur. Serveri i mikrokontrollorit (mbyllur) është i pavaruar nga teknologjia, dhe gjenerohet në gjuhën asembler. Karakteristika të kodit të gjeneruar janë madhësia shumë e vogël reale e kodit të skedarit HEX që ka për tu shkruar në kujtesën programuese të mikrokontrollorit, dhe madhësia shumë e vogël e mesazheve në mes serverit lidhës dhe atij të mbyllur, për hyrje të ndryshme të gjeneratorit të kodit. Hyrjet e gjeneratorit janë skedarët IDL (ang. Interface Definition Language) dhe ai implementues. Kjo falë natyrës së adaptueshme të protokollit AMISP, pra gjenerimit të fushave që janë vetëm të domosdoshme për identifikim të operacionit për tu ekzekutuar.

Në fund të këtij punimi është demonstruar një sistem i monitorimit të jetëgjatësisë së baterive, për çka sistemet e tilla rëndom shërbejnë – marrje të dhënash dhe dirigjim.

ABSTRACT

This work deals with the integration of embedded devices into systems based on Service Oriented Architectures (SOA). Integration implies the implementation of SOA-based technologies, such as CORBA and Web Services, on embedded devices. This simply means that the embedded device executes the code that would make it a CORBA Service or Web Service, that would respond to client requests. Though for powerful devices such as personal computers this can be considered a work done, the integration of embedded devices remains a challenge. This is because such service-oriented technologies require plenty of resources like program memory and RAM, which are scarce in embedded devices. In this way, they can not be realized in embedded devices such as microcontrollers. The role of the microcontroller is to exchange data with a powerful device such as a personal computer, for example, accepting commands from the computer and applying the respective signals to the sensors or actuators connected to the microcontroller. Building SOA based system enables further system expansion and maintenance in a standardized way. In the literature the common and compressed approach is applied. In the common approach, there is no compiler for translating the code generated by the corresponding tools such as gSOAP to executable 8051 code. Compressed approach generates excessive bit-manipulation code such as bit shifting, comparing and applying logical operations. In either case, the resulting code is too large to be programmed into the too-small memory of the 12 kilobytes of the 8051 microcontroller we use. This makes both approaches inadequate and we propose a new approach with significantly smaller resources. The new approach is based on sharing responsibilities between powerful devices such as a personal computer and a microcontroller.

We propose the solution by introducing the bind server concept. The bind server runs on powerful devices like a personal computer, and deals with complex jobs, leaving the microcontroller to deal with the simple ones. The client only sees the bind server, which translates the high level requests from the client into a simple ones and by forwarding them to the server in microcontroller. The client and the bind

server communicate through the technology specific protocol for which are implemented. Thus, for CORBA technology they communicate based on the IIOP (Inter-Orb Protocol) protocol, while for Web Services communicate through Simple Object Access Protocol (SOAP). On the other hand, the bind server and the embedded device (microcontroller) communicate through the protocol we have designed, called the AMISP (Adaptive Minimal Inter Server Protocol). So, the communication between the bind server and the embedded server is not dependent on the client and bind server implementation technologies. In this way, it is possible to have the embedded server at the same time accessed by different technologies. All this work done manually would have been exposed to mistakes, time consuming, and detailed technical knowledge would be needed. Thus, we have developed code generators that would generate the bind server and embedded device code. The bind server is completely generated and depends on the technology, e.g. is another for JacORB (implementation in CORBA's Java) and another for JAX-WS (implementation in Java for Web Services). Indeed, for the given programming language they are very similar and can be transformed from one to the other by modifying only a few lines of source code. For the embedded device server, only the communication with the bind server is generated, with whom commonly exchanges the operation parameters. What remains to be written by the developer is the implementation of the operations for which they are intended. The microcontroller (embedded) server is independent of the technology, and is generated in the assembler language. Characteristics of the generated code are the very small real size of the HEX file code that is to be written in the microcontroller memory, and the very small size of the messages between the bind server and the embedded server, for different inputs of the code generator. Generator inputs are IDL (Interface Definition Language) and implementation files. This is due to the adaptive nature of the AMISP protocol, i.e. the generation of fields that are only necessary to identify the operation to be executed.

At the end of this work a battery life monitoring system has been demonstrated, for which such systems usually serve - data acquisition and control.

FJALOR TERMINOLOGJIK

A. O. Sh. (Arkitekturë e Orientuar në Shërbime) – Arkitekturë e Orientuar në Shërbime është stil i dizajnit të softuerit ku shërbimet u ofrohen komponenteve tjera nga komponente të aplikacionit, përmes protokollit komunikues nëpër rrjet. Principet themelore të arkitekturës së orientuar në sherbime janë pavarësia nga prodhuesi, produkti dhe teknologjia.

AMISP (Adaptive Minimal Inter-Server Protocol) – Protokoll që shërben për komunikim në mes serverit lidhës dhe serverit në mikrokontrollor.

Coco/R – Është gjenerator i kompilatorit, që merr si hyrje gramatikën e atribuuar të gjuhës burim dhe gjeneron skanerin dhe parserin për atë gjuhë.

CORBA (Common Object Request Broker Architecture) – Është standard i përcaktuar nga Object Management Group i dizajnuar për të lehtësuar komunikimin e sistemeve nga platforma të ndryshme.

Gjenerator kodi – Quhet vegla që për të dhënat si hyrje në gjuhën e caktuar gjeneron të dhënat si dalje në gjuhën tjetër të caktuar.

IDL (Interface Definition Language) – Gjuha e përkufizimit të ndërfaqes është gjuhë specifikuese e përdorur për përshkrim të ndërfaqeve të shërbimeve të bazuar në CORBA. IDL-të e përshkruajnë ndërfaqen në mënyrë të pavarur nga gjuha programuese e implementimit të shërbimit.

JacORB – Është projekt më kod të hapuar në gjuhën programuese Java për krijim të shërbimeve të bazuar në CORBA.

JAX-WS (Java API for XML Web Services) – Është API në gjuhën programuese Java për krijim të shërbimeve ueb.

Mikrokontrollor – Mikrokontrollori është kompjuter i vogël në një qark të vetëm të integruar.

Ndërfaqe (ang. Interface) – Është kufiri i përbashkët në të cilin dy ose më shumë komponente të ndara të një sistemi kompjuterik shkëmbejnë informata.

Pajisje e mbyllur (ang. Embedded device) – Pajisje e mbyllur është pajisje e specializuar e dedikuar për një ose disa qëllime specifike dhe rëndom është pjesë e ndonjë objekti tjetër ose sistemi të gjerë.

Portë seriale (ang. Serial port) – Lidhëse me ndihmën e së cilës dërgohet ose pranohet një bit në kohën e dhënë.

Server lidhës – Server që ndërmjetëson ndërmjet klientit dhe serverit në mikrokontrollor.

Server në mikrokontrollor – Server që ndodhet dhe ekzekutohet në mikrokontrollor.

Shërbim Ueb (ang. Web Service) – Shërbimi ueb është shërbim i ofruar pajisjes elektronike nga pajisja tjetër elektronike, duke komunikuar njëra me tjetrën përmes uebit.

Signature – Emërtimi i metodës bashkë me numrin, tipat dhe rradhën e parametrave të saj.

SPI (Serial Peripheral Interface) – Është ndërfaqe bus e përdorur zakonisht për të shkëmbyer të dhëna ndërmjet mikrokontrollorëve dhe sensorëve, regjistrave shift, etj.

Stream – Seri e bajtëve që shkëmbehet për komunikim ndërmjet serverit lidhës dhe mikrokontrollorit ku bajtë të vetëm ose grupe bajtësh përmbajnë fusha identifikuese ose vlera parametrash të operacioneve (metodave).

WSDL (Web Service Description Language) – Gjuha e përshkrimit të shërbimit ueb është gjuhë e përkufizimit të ndërfaqes e bazuar në XML (ang. eXtensible Markup Language), që përdoret për përshkrim të funksionalitetit të ofruar nga shërbimi ueb.

KAPITULLI 1

Hyrje dhe motivim

1.1 Hyrje

Sistemet e sotme të marrjes së të dhënave dhe të dirigjimit mund të jenë shumë të shpërndara në kuptim të vendit në hapësirë dhe shumë komplekse në kuptim të numrit të pajisjeve (nyjeve) që përbëjnë sistemin. Ndërtimi i sistemeve të tilla mund të jetë mjaftë i vështirë. Krahas përdorimit të teknologjive të dedikuara (softuerit të specializuar) kërkohet edhe angazhim i njerëzve të trajnuar mirë dhe profesional për implementim të tyre. Kjo është e kushtueshme, por siguron zgjerim të mëtejshëm të sistemit dhe përveç tjerash performansë dhe kualitet. Në të kundërtën, ndërtimi i sistemeve në mënyrë të jo-standardizuar d.m.th. *ad-hoc* ka përparësi kur sistemi është i vogël me disa burime të dhënash dhe ku nuk kërkohet shpërndarje ose rrjetëzim, por mund të shpie në mangësi serioze siç është vështirësimi i zgjerimit të mëtejshëm të sistemit dhe shkallëzimi. Ne inkurajojmë ndërtimin e sistemeve me softuer të specializuar si rrugë e vetme për të bërë punën si duhet. Kur konsiderojmë softuerin paralelisht duhet të konsiderojmë po ashtu edhe pajisjet në të cilat ai softuer do të ekzekutohet. Duke i konsideruar më tej edhe trendet ekonomike në të cilat jetojmë dhe çmimin e pajisjeve të tilla të specializuara, ne duhet të paraqesim ide të reja që do mundësojnë ndërtimin e sistemeve të marrjes së të dhënave dhe dirigjimit me pajisje konvencionale, më të rëndomta, që mund të gjenden në tregun tonë. Kjo duhet të vlejë edhe për softuerin. Mundësisht, duhet të përdorim softuer pa pagesë dhe me kod të hapur me qëllim të modifikimit të tij që t'ia përshtatim nevojave tona dhe ta përdorim lirshëm pa kufizim. Kjo mundëson që sistemet e tilla të mund të ndërtohen nga organizata me financa të mangëta siç janë Universitetet dhe institucionet tjera publike.

1.2 Motivimi

Mikrokontrollorët [3][4][5] janë të përdorur gjerësisht për të komunikuar me sensorët dhe aktuatorët në konfiguracion me PC [17][18]. Kjo është për shkak të

çmimit shumë të ulët në krahasim me pajisjet tjera që kryejnë punë të njëjtë me performansë të ngjashme. Po ashtu PC-të janë mjaftë të pranishëm në treg, dhe ne synojmë të i inkorporojmë ata standardë për shkak se në shumë raste i kënaqin performansat e kërkuara. Siç e dijmë, sistemet për marrje të dhënash dhe dirigjim mund të mos jenë intensive në përpunim të dhënash siç janë p.sh. aplikacionet që kanë të bëjnë me grafikë tri dimensionale. Sidoqoftë, kërkesat për kohë reale, mostrimi në frekuenca të larta dhe transferimi i të dhënave janë kërkesa vendimtare në zgjedhjen e komponenteve të PC-së.

Skema e një konfigurimi mikrokontrollor-PC është paraqitur në figurën 1.1.

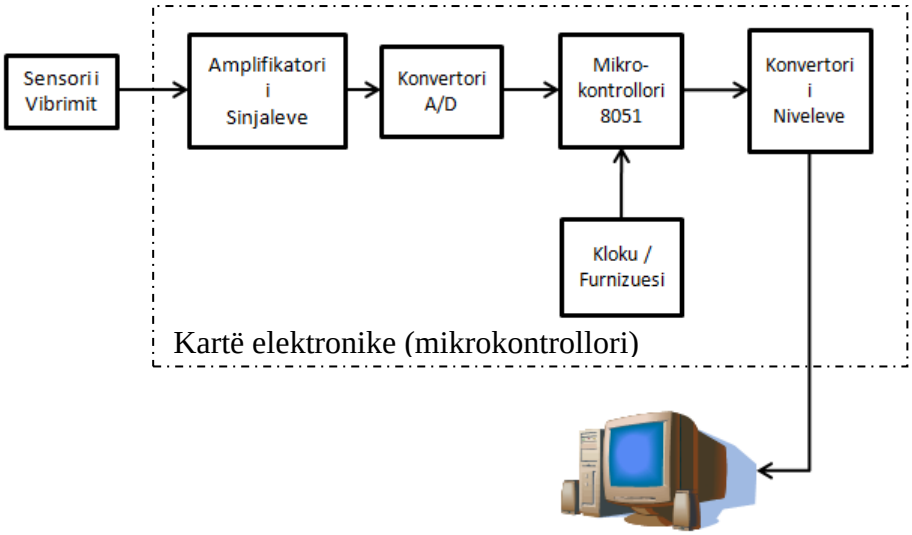


Figura 1.1 Mikrokontrollori në konfiguracion me PC për marrje të dhënash (ri-konstruktuar nga [18])

Në të vërtetë, krahas mikrokontrollorit si çip, aty janë edhe komponentet tjera (më pak të rëndësishme për këtë punim) të nevojshme dhe mund të gjenden në një kartë të vetme si në [19].

Mikrokontrollori shërben për përkthim të të dhënave në nivel të lartë (bajtëve) që vijnë nga PC në nivel të ulët (sinjale) që kanë si destinacion sensorin apo aktuatorin, ose e kundërta – siç është treguar në figurë nga sensori në PC. Në fakt, në figurën 1.1 shigjetat tregojnë që shënimet rrjedhin prej mikrokontrollorit në PC, por, në përgjithësi drejtimi i kundërt është rëndom i mundshëm. Ata komunikojnë përmes ndërfaqes seriale RS-232 që mund të gjendet i integruar në shumë mikrokontrollorë.

Edhe pse komunikimi serial është i integruar në mikrokontrollorë, mund të gjenden karta [20] me porte për ethernet por kjo ka si pasojë çmimin.

1.3 Kontributi i këtij punimi në terma të përgjithshëm

Arkitekturat e dy projekteve [17][18] përdorin mikrokontrollorin d.m.th. e integrojnë dhe komunikojnë me të në mënyrë specifike të projektuar për projekt. Projekti [17] është më i veçantë pasi që mikrokontrollori është i lidhur edhe me pajisje tjera përveç sensorëve dhe aktuatorëve dhe duhet të menaxhohet prej distance – për dallim nga mënyra standarde, përmes portës së tij seriale të nënkuptuar. Nga aspekti jonë i vështrimit projekti [18] është më i rëndomtë, për arsye të infrastrukturës së lidhjeve dhe komponenteve të përfshira. Ne nuk merremi me aplikimin (qëllimin) e projekteve të lartpërmendura, por, vetëm me komunikimin dhe integrimin e mikrokontrollorit. Çka u mungon këtyre punimeve është standardizimi i komunikimit. Pra, mikrokontrollori komunikohet përmes protokollit të ndërtuar ad-hoc sipas nevojave të detyrës ose aplikimit. Me fjalë të tjera, u mungon mënyra e unifikuar e komunikimit (shkëmbimit të të dhënave) me mikrokontrollorin.

Ne synojmë të ndërtojmë mënyrë të standardizuar, të konfiguruar, të shkallëzuar dhe lehtë të mirëmbajtur të komunikimit me mikrokontrollor. Në përgjithësi, ne e shohim mikrokontrollorin si komponente të rrjetës, që do të mund të integrohej dhe qasej edhe përmes Internetit krahas prej LAN (ang. Local Area Network). Skema e përgjithësuar e një konfigurimi të tillë do të mund të paraqitej si në figurën 1.2.

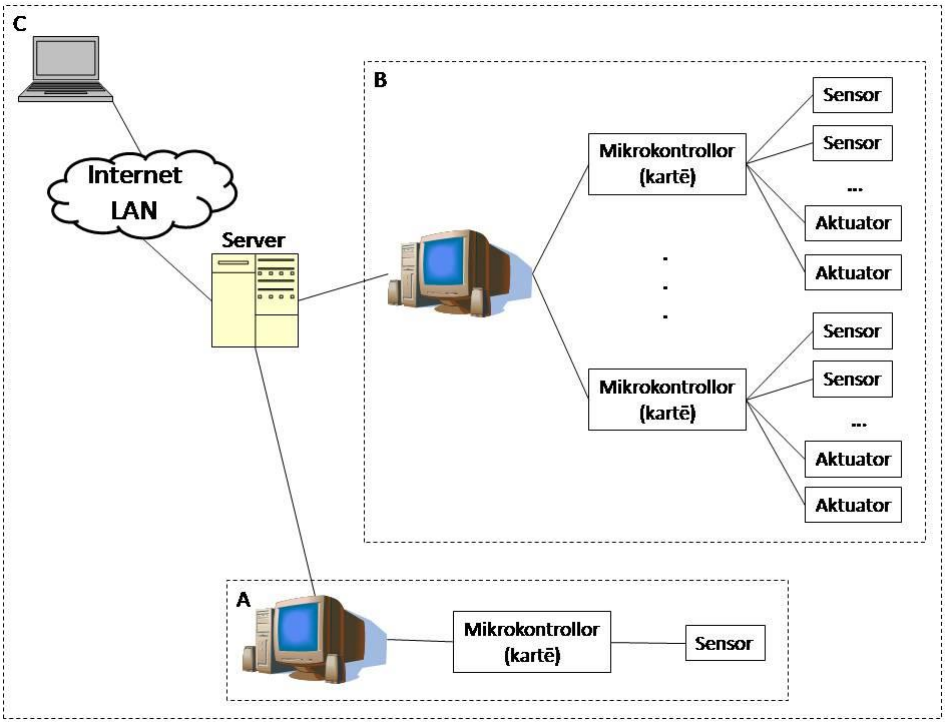


Figura 1.2 Skema e përgjithësuar e konfiguracionit të mikrokontrollorit

Në figurën 1.2 është paraqitur skema e përgjithësuar e konfiguracionit të mikrokontrollorit në sistem për marrje të dhënash dhe dirigjim. Siç mund të shihet, sipërfaqja 'A' është konfiguracioni i rëndomtë siç është paraqitur po ashtu edhe në [18]. Ne e zgjerojmë këtë më tej si në 'B', kur më shumë mikrokontrollorë janë të lidhur për PC, dhe mikrokontrollorët marrin të dhëna nga më shumë sensorë dhe ngasin më shumë aktuatorë. Në vështrim të parë, ky duket si problem i rëndomtë, dhe shumë mikrokontrollorë për PC mund të zgjidhet duke e ndarë problemin "shumë" në "një", duke menduar bashkangjitjen e secilit mikrokontrollor për një portë. Pastaj, prezenca e shumë sensorëve dhe aktuatorëve për mikrokontrollor është e rëndësisë së veçantë për të iu shmangur kompleksitetit dhe për ti shfrytëzuar burimet e mikrokontrollorit optimalisht. Të dyja kanë rol të rëndësishëm në zgjidhjen që do ta ofrojmë më vonë duke paraqitur protokollin tonë të ri. Më tej, ne e zgjerojmë 'B' sikur në 'C', ashtu që tani mikrokontrollori do të mund të qasej nga rrjeti lokal ose në përgjithësi edhe prej Internetit. Kjo sipas logjikës bën pyetjen a duhet të ndërtohet platformë e posaçme (dedikuar) ngjashëm me [21] për ti kënaqur kërkesat e arkitekturës sikur në sipërfaqen 'C', ose të përdorim teknologjitë e rëndomta ueb të lëmisë siç janë Shërbimet Ueb [22], CORBA [23], .NET Remoting [24] dhe të

ngjashme. Ky nuk është problem i ri në shkencë, dhe zgjidhja jonë nuk është tërësisht si në të parën (teknologji e posaçme) ose si në të dytën (teknologji e rëndomtë), por, siç do të shohim, kompromis në mes të dyve.

Ndërtimi i platformës së veçantë kushton në kohë dhe është jashtë objektivave tona. Do të përfitonim në performansa duke konsideruar pajisjet e limituara që kemi për synim, por, do të i bënte ato jo-kompatibile dhe do të kërkonte trajnim shtesë të njerëzve që do ta përdornin, që implikon kosto dhe shpenzim kohe. Në anën tjetër, përdorimi i teknologjive të rëndomta ueb, do ta bënte sistemin tonë më të lehtë për të u zhvilluar, sepse ka më shumë njerëz që kanë njohuri mbi to, dhe do ta bënte të vendosej lehtë në rrjet ose Internet sepse janë të dedikuara për to, por, janë vështirë të implementohen në pajisje të limituara (ose primitive) siç janë mikrokontrollorët për shkak të hapësirës së madhe të kujtesës dhe kodit që zënë. Punimi [25] pohon që mikrokontrollorët 8051 [3], krahas të tjerëve, nuk e kanë fuqinë llogaritëse për të përkrahur infrastrukturën e kërkuar për të implementuar shërbimet ueb, kështu pajisjet me shërbime ueb do të ndërtoheshin mbi procesorë më të "mëdhenj" ose variante më të "mëdha" të mikrokontrollorëve. Pavarësisht komenteve të paraqitura në [25] ose punime tjera përkatëse, ka punime të ndryshme që japin zgjidhje lidhur me problemin tonë d.m.th. integrimin e pajisjeve primitive në përgjithësi (dhe të mikrokontrollorëve në veçanti) në teknologjitë e rëndomta ueb. Duhet theksuar se punimet në fjalë e adresojnë problemin për pajisje të mbyllura shumë më të fuqishme se mikrokontrollori 8051. Mikrokontrollori 8051 mbetet akoma sfidë. Përdorimin e tij e arsyeton çmimi dhe gjerësia e përdorimit. Duhet të theksohet edhe se ky punim ka qëllim integrimin në AOSh të pajisjeve jo të gatshme për AOSh. Pajisjet jo të gatshme për AOSh dallohen nga ato të gatshme për AOSh sepse nuk posedojnë pajisje për lidhje në rrjeta standarde (TCP/IP) si kartë rrjeti dhe stivë komunikimi. Në kapitullin vijues i paraqesim disa nga zgjidhjet që i kemi gjetur gjatë kërkimit tonë për literaturë.

1.4 Organizimi i punimit

Ky punim është i organizuar si vijon. Në kapitullin **1 Hyrje dhe motivim** është paraqitur njoftim me problemin që shtjellohet në këtë tezë. Në pika të shkurtra është shkruar mbi problemin, motivimin si dhe kontributin në përgjithësi.

Në kapitullin **2 Të arriturat relevante në literaturë** janë paraqitur të arriturat në literaturë që kanë të bëjnë me problemin e parashtruar në tezë, duke i diskutuar përparësitë dhe anët negative.

Në kapitullin **3 Propozimi për kontribut në tezë** është dhënë propozimi për kontribut në këtë disertacion. Propozimi bazohet në analizat e literaturës relevante dhe trendet më të reja të lëmisë siç janë Arkitekturat e Orientuara në Shërbime.

Në kapitullin **4 Teknologjitë e involvuara në hulumtim** janë paraqitur shkurtimisht detaje të teknologjive që hyjnë në punë në këtë punim. Është paraqitur mikrokontrollori 8051, vegla kompilatorike Coco/R si dhe teknologjia e orientuar në shërbime JacORB.

Në kapitullin **5 Realizimet – strukturat e kodit dhe gjeneratorët** janë paraqitur realizimet teknike e kësaj teze. Ato janë: gjeneratorët e kodit për integrimin e pajisjeve të mbyllura në AOSh, struktura e kodit të gjeneruar dhe klasat ndihmëse të zhvilluara që nevojiten për punë.

Në kapitullin **6 Rezultatet – madhësitë e skedarëve, mesazheve dhe kohët e komunikimit** janë paraqitur rezultatet e kontributit që maten me madhësi reale të kodit të mikrokontrollorit dhe madhësi të mesazheve të shkëmbyer ndërmjet serverit lidhës dhe atij në mikrokontrollor. Poashtu, janë paraqitur edhe kohët e komunikimit në mes klientit dhe serverit në mikrokontrollor, duke i krahasuar me rezultatet nga literatura.

Në kapitullin **7 Shembull – ndërtimi i një sistemi për marrje të dhënash** janë paraqitur detaje të një sistemi të monitorimit të jetëgjatësisë së baterive, me veglat gjenerator kodi që janë zhvilluar në kuadër të kësaj teze.

Në fund, në kapitullin **8 Përfundime dhe puna për të ardhmen** janë paraqitur përfundimet e kontributit të kësaj teze, si dhe puna për të ardhmen për të avansuar me të arriturat e lëmisë.

KAPITULLI 2

Të arriturat relevante në literaturë

2.1 Hyrje

Në këtë kapitull do të paraqiten të arriturat relevante lidhur me integrimin e pajisjeve primitive të mbyllura në arkitektura të orientuara në shërbime (AOSh). Në shumicën prej tyre platformë konkrete AOSh janë Shërbimet Ueb (ang. Web Services). Janë dhënë edhe detaje të protokolleve më të përdorshëm të lëmisë.

2.2 Analizë e të arriturave

Punimi [11] paraqet platformën virtuale hibride për simulim të sistemeve të distribuuara të pajisjeve të mbyllura. Platforma quhet hibride sepse në të mund të integrohen pajisje reale elektronike si dhe modele ekuivalente softuerike të tyre. Mirëpo, fokusi është përqendruar në ato të modeluara softuerike, me qëllim që të sajohen situata që mund të ndodhin rrallë realisht. Pjesët përberëse të sistemit (simulatori i njësisë qendrore përpunuese dhe simulatori i pajisjes) komunikojnë ndërmjet veti përmes teknologjisë së distribuuar CORBA. Shkëmbimi i mesazheve behët me shpejtësi të krahasueshme me atë të realizuar me pajisje reale. Sa i përket komunikimit me CORBA, vetëm një nënbashkësi e llojeve të mesazheve është implementuar, edhe atë mesazhet: *request*, *reply*, *locaterequest* dhe *locatereply*. Komunikimi me pajisje reale bëhet përmes *gateway*. *Gateway* e merr mesazhin nga klienti, e veçon segmentin e mesazhit nga mesazhi CORBA dhe ia dërgon pajisjes reale përmes rrjetës reale CAN (ang. Controller Area Network). Poashtu, e merr mesazhin përgjigje nga pajisja reale përmes rrjetës reale (CAN) dhe ia dergon klientit që i korrespondon mesazhit kërkesë përkatës të dërguar më parë nga klienti.

Punimi [12] merret me implementimin e AOSh në pajisje me burime të kufizuara te të cilat nuk mund ekzekutohet stiva TCP/IP dhe kështu nuk mund të integrohen në menyrë të drejtperdrejtë në rrjeta të bazuara në IP. Zgjidhja konsiston

në përdorimin e logjikës së funksionimit të protokollit fjalëshumë HTTP në vend të përdorimit të vetë HTTP-së. Qasja rrjetit WSN (ang. Wireless Sensor Network) nga rrjeti standard i Internetit realizohet përmes shtresave të cilat kanë për qëllim primar "përkthimin" e kërkesave fjalëshumë HTTP në ato të shkurta µIP. Arkitektura AOSh katër shtresore duket si në figurën 2.1.

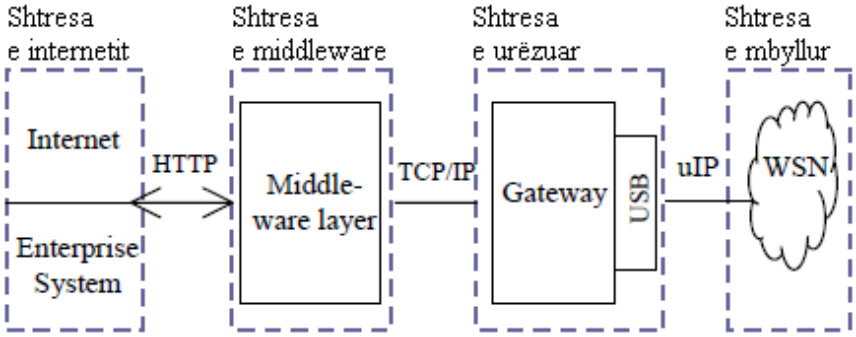


Figura 2.1 Shtresat e arkitekturës AOSh (sipas [12])

Shtresa e mbyllur konsiston në vetë pajisjet e mbyllura të kufizuara. Pajisjet i përgjigjen kërkesave të inicuara nga shtresa e Internetit dhe të "përkthyera" nga shtresa *middleware* përmes formatit JSON standard apo atij të komprimuar të zhvilluar posaçërisht për këtë rast. Shtresa e urëzuar konsiston në pajisjen që transferon kërkesat nga TCP/IP në atë ZigBee të destinuar dhe pranueshme për shtresën e mbyllur. Shtresa *middleware* që praktikisht paraqet një PC ka funksione të ndryshme. Përveç "përkthimit" të kërkesave fjalëshumë HTTP në ato JSON (ose JSON të komprimuara), aty ruhen informatat mbi adresat e pajisjeve në rrjetin WSN si dhe informata tjera shtesë.

Punimi [13] paraqet një nga arritjet e hershme mbi integrimin e pajisjeve të mbyllura përbërëse të WSN në rrjetat IP të rëndomta. Autorët propozojnë protokollin TinyREST, që është rezultat i kombinimit të sistemit të mesazhimit të integruar në sistemin operativ TinyOS mbi të cilin punojnë pajisjet e mbyllura MicaZ, dhe HTTP protokollit tanimë të njohur REST. Klientët inicojnë kërkesa REST të cilat përmes HTTP-2-TinyRest gateway përkthehen në TinyREST dhe i transmetohen pajisjeve që kontrollojnë sensorët/aktuatorët pa tela. Kërkesat REST e kanë formën "METODA URL", ku METODA mund të jetë përveç tjerash GET ose POST që shërben respektivisht për leximin e sensorit dhe kontrollin e aktuatorit. URL paraqet adresën e

pajisjes së mbyllur, dhe mund të jetë URL tipik si /gatewayIP/floor1/light1. Siç theksuam, këto kërkesa përkthehen në mesazh TinyREST përmes pasqyrimeve që janë ruajtur në HTTP-2-TinyRest gateway duke i shoqëruar kështu çdo nyeje (pajisje) të WSN dy numra unik atë të grupit që i takon pajisja dhe vetë pajisjes. Madhësia e mesazhit TinyREST në pajisjen MicaZ është e kufizuar nga stiva TinyOS i rrjetit në 29 bajtë.

Punimi [14] paraqet veglën gSOAP, gjeneratorin e kodit në gjuhën C/C++ për realizim të komunikimit bazuar në protokollin SOAP (ang. Simple Object Access Protocol) të njohur si Shërbime Ueb (ang. Web Services). gSOAP gjeneron shtyllat dhe skeletet që u mundëson klientëvë dhe serverëve të komunikojnë në mes tyre. Madhësia e vogël e kodit (pas kompilimit) të gjeneruar dhe shpejtësia e komunikimit e ka bërë këtë vegël të përshtatshme për përdorim në pajisje të mbyllura. Kjo falë algoritmeve të përparuara të (de)marshallimit dhe prekompilimit të shtyllave dhe skeleteve. Megjithatë, janë raportuar performansa më të dobëta në krahasim me teknologjitë konkurente si CORBA, që përdor protokoll binar në krahasim me atë fjalëshumë SOAP të bazuar në XML (ang. eXtensible Markup Language). Nga ana tjetër SOAP është më i thjeshtë, i lexueshëm dhe më i afërt me përdoruesin.

Punimi [15] paraqet një zgjidhje për integrim të pajisjeve të mbyllura të kufizuara në burime dhe fuqi në Shërbime Ueb. Zgjidhja konsiston në arkitekturën dy shtresore –atë të bazuar në SOAP dhe atë në REST (figura 2.2). Shtresa e epërme, nyja me funksionalitet të plotë është e qasshme përmes protokollit SOAP nga aplikacionet klient nga jashtë, dhe njëkohësisht paraqitet si klient për shtresën e poshtme, nyjen me funksionalitet të reduktuar të implementuar si Shërbim Ueb REST. Nyja me funksionalitet të plotë është e realizuar në kompjuter personal dhe iu është e ekspozuar rrjetit të jashtëm (Internetit). Shërben për funksione komplekse që nuk janë të përshtatshme të realizohen në pajisje të mbyllur siç janë: qasja konkurrente nga më shumë se një shfrytëzues, logim i qasjeve, komunikimi i enkriptuar etj. Nyja me funksionalitet të plotë mund t'i qaset disa nyjeve me funksionalitet të reduktuar. Nyja me funksionalitet të reduktuar e realizuar si Shërbim Ueb REST implementohet në pajisje të mbyllur për shkak të thjeshtësisë së protokollit REST. Nyjet me funksionalitet të reduktuar janë të lidhura në rrjet privat, të cilave mund ti qaset vetëm

nyja me funksionalitet të plotë. Nëpërmjet këtij rrjeti mund të lexohen vlerat e parametrave të caktuar nga sensorët ose t'u vendosen vlera parametrave të caktuar aktuatorëve. Me këtë rast supozohet se komunikimi nuk është intensiv dhe i shpejtë, por lexim/vendosje e vlerave të parametrave kohë pas kohe.

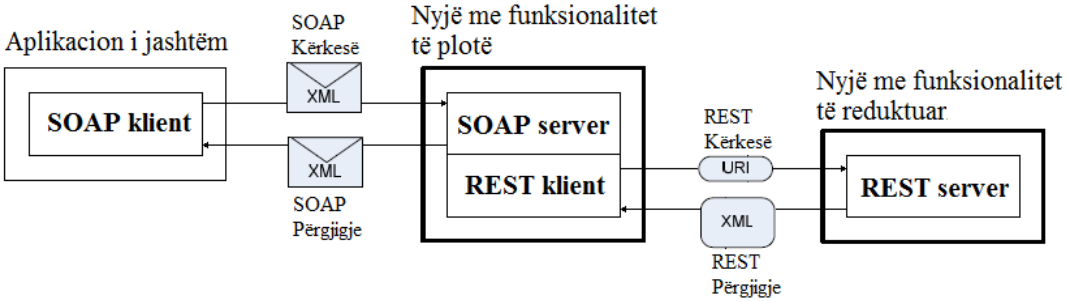


Figura 2.2 Arkitektura dy shtresore SOAP/REST (rikonstruktuar nga [15])

Punimi [16] paraqet alternativë të implementimit të Shërbimeve Ueb në pajisje shumë të limituara përmes qasjes së udhëhequr në mënyrë tabelare. Fillimisht ndërtohet servisi i rëndomtë sipas specifikimeve të DPWS (ang. Device Profile for Web Services). Pastaj, në kompjuter analizohen të gjitha mesazhet e shkëmbyera ndërmjet klientit dhe servisit të implementuar. Nga mesazhet e shkëmbyera konkludohet për pjesët e mesazhit SOAP të cilët përdoren më së shpeshti. Këto pjesë statike zëvendësohen me stringje përkatëse. Kështu, mesazhet korrespondente ndërtohen duke bashkuar stringjet që nuk ndryshojnë dhe pjesës që ndryshon në mesazh, në vend të etiketave (ang. tags) XML të mesazhit SOAP. Kjo qasje ka përparësi pasi nga analizat e mesazheve të shkëmbyera konkludohet se gati gjithmonë shkëmbehen mesazhe të njëjta, vetëm 4,6 % e përmbajtjes së mesazheve të shkëmbyera ndryshon. Nga ana tjetër zëvendësimi i XML me stringje të thjeshta e lehtëson parsimin e mesazheve në krahasim me alternativën SOAP. Madje, duke e ditur paraprakisht gjatësinë e stringjeve paraqitet mundësia që komunikimi të bëhet edhe i parashikueshëm në kohë.

Punime interesante rreth integrimit të pajisjeve të mbyllura në arkitektura të orientuara në shërbime janë paraqitur nga Käbisch dhe të tjerët [40][41][42]. Këto punime janë interesante për ne pasi që paraqesin integrimin e mikrokontrollorëve si pajisje mjaftë të kufizuara në Shërbime Ueb. E tërë puna bazohet në standardin EXI [27] që paraqet alternativën binare të standardit tanimë të popullarizuar XML. Në

vend të shkëmbimit të mesazheve SOAP ndërmjet klientëve dhe serverëve të realizuar në teknologjinë e Shërbimeve Ueb, ata shkëmbejnë mesazhe EXI që ndërtohen duke koduar alternativat përkatëse të mesazheve SOAP. Derisa punimi [40] paraqet punën fillestare që është gjenerimi i dispeçerit, procesorit EXI dhe skeletëve RPC (ang. Remote Procedure Call), punimet [41] dhe [42] paraqesin përparime dhe optimizime në atë drejtim. Në të vërtetë, gjenerimi i komponenteve të përmendura bëhet në dy faza. Në fazën e parë, bazuar në përshkrimin e servisit me WSDL (ang. Web Service Description Language) gjenerohen shkemat që paraqesin të gjitha mesazhet e mundshme që shkëmbejnë klienti dhe serveri. Në fazën e dytë gjenerohen procesori EXI, skeleti RPC dhe dispeçeri. Procesori EXI shërben për kodim/dekodim të mesazheve nga/në strukturat e të dhënave. Skeletet RPC përmbajnë implementimet e operacioneve në mikrokontrollor nga zhvilluesi i aplikacionit. Dispeçeri merr bajtët EXI, ekzekuton RPC dhe kthen përgjigjen në bajtë EXI. Kodimi i mesazheve SOAP në EXI bëhet ashtu që çdo instance të mesazhit SOAP i përgjigjet një vlerë binare e koduar. Pasi që XML (në bazë të së cilës ndërtohet SOAP) nuk është gjuhë atëherë përdoret stiva e gramatikave për përshkrim të entiteteve të ndryshme. Përparësi e transformimit të skemave XML në gramatika qëndron në atë se gramatikat janë më të thjeshta për tu përpunuar. Gramatikat EXI realizohen si automata të fundme deterministike. Optimizimet kanë të bëjnë me mënjanimin e pjesëve të tepërta siç është p.sh. eliminimi i mesazheve të tjera shtesë që mund të përdoren nga Shërbimet Ueb dhe implementimi i vetëm kërkesave/përgjigjeve (operacioneve) që shkëmbehen ndërmjet klientit dhe serverit. Poashtu ka reduktim të kodit që në server implementohet vetëm kodi për dekodim të kërkesave, ekzekutim të operacionit dhe kodim të përgjigjeve. Ngjashëm klienti implementon kodin për kodim të kërkesave, thirrje të operacionit dhe dekodim të përgjigjes. Rezultatet e këtyre punimeve maten më madhësi të kodit rezultues, performansa dhe madhësi të mesazheve të shkëmbyera. Kështu mesazhet e shkëmbyera janë rreth 80 herë më të vogla se alternativa standarde XML e tyre, performansa e shpejtësisë së shkëmbimit është rreth 4 herë më e shpejtë se alternativa XML e realizuar përmes versionit ekuivalent gSOAP [14], ndërsa madhësia e kodit rezultues është rreth 6 herë më e vogël se ajo e implementuar si gSOAP.

EXI [27] është komprimim binar i informatës së paraqitur në formë të XML. Pasi që në Shërbime Ueb SOAP klienti dhe serveri shkëmbejnë mesazhe bazuar në XML (SOAP), atëherë për pajisje të mbyllura (të bazuar në SOAP), për ta reduktuar trafikun aplikohet EXI për shkëmbim të mesazheve. EXI funksionon bazuar në gramatika. Ai përdor numër të caktuar të bitëve për të identifikuar opsionin (operacionin, zgjedhjen e elementit XML për përpunim). Kështu nëse janë *n* opsione atëherë numri i bitëve të nevojshëm për ta përcaktuar opsionin e zgjedhur është $\lceil \log_2 n \rceil$. Për ta identifikuar strukturën dhe ndërtimin (kodim/dekodim) e mesazheve të shkëmbyera procesorit EXI i duhet skema XML e përshkrimit të funksionalitetit të serverit. Skema e përshkrimit të funksionaliteteve është e njohur si skedari WSDL (ang. Web Service Description Language). Skema është e nevojshme në të dyja anët, në të klientit dhe atë të serverit. Nëse struktura e dokumentit XML (mesazhit) është komplekse, pra nëse brenda një elementi (opsioni) mund të paraqiten elemente (opsione) të tjerë, atëherë përdoret stiva e gramatikave. Kur hyhet në përpunim të elementit bëhet push i gramatikës për atë element (i cili mund të ketë nën elemente të tjera) dhe kur dilet nga ai element behet pop i gramatikës për atë element. Pasi gramatikat punojnë si automata të fundme deterministike, prej aty edhe nevoja për numër të caktuar të bitëve si hyrje për kalim nga një gjendje në tjetrën.

CoAP [28] paraqet alternativën binare të realizimit të Shërbimeve Ueb të bazuar në protokollin HTTP REST (ang. REpresentational State Transfer). Është i dedikuar për pajisje të kufizuara në fuqi dhe burime siç janë mikrokontrollorët 8 bitësh. Siç dihet REST përdor metodat HTTP GET dhe POST, të cilat CoAP i transformon në formë binare për operacionet e dhëna të përcaktuara në skedarët përshkrues të shërbimit WADL (ang. Web Application Description Language). Madhësia e mesazhit është e vogël, për ta bërë kështu të përshtatshëm për t'u transferuar në rrjet përmes 6LoWPAN si pako UDP. Specifikacioni CoAP përcakton katër lloje të mesazheve: *Confirmable*, *Non-confirmable*, *Acknowledgement* dhe *Reset*. Kërkesat mund të barten në mesazhet *Confirmable* dhe *Non-confirmable*, dhe përgjigjet në këto dy dhe poashtu në mesazhet *Acknowledgement*. Formati i mesazhit CoAP është dhënë në figurën 2.3. Siç mund të shihet fushat përcaktohen në grupe bitësh në kuadër të një ose më tepër bajtëve. Kjo do të paraqiste punë shtesë në

mikrokontrollor për krahasim të tyre pasi që do të implikonte operacione të zhvendosjes (ang. shift) së bitëve për formim të vlerës së caktuar.

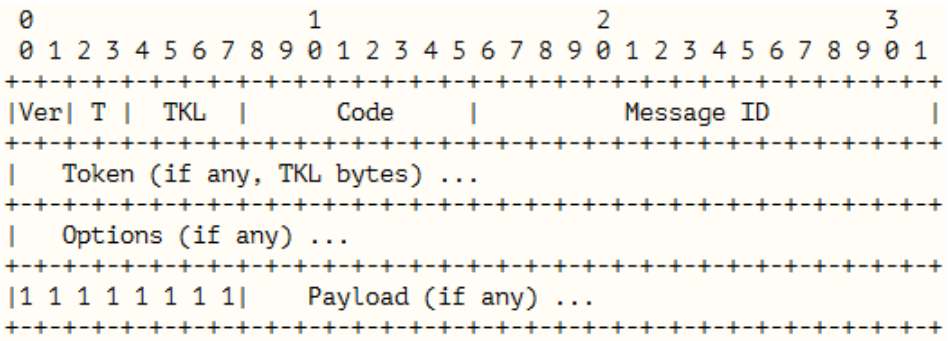


Figura 2.3 Formati i mesazhit CoAP (sipas [28])

Modbus [43] është njëri nga protokollet që është gjatë në përdorim. Në fillim ka qenë i projektuar për përdorim në sisteme të kontrollit dhe marrjes së të dhënave të bazuar në PLC (ang. Programmable Logic Controller) i njohur si Modbus RTU (ang. Remote Terminal Unit). Në këtë rast komunikimi ka qenë serial [44] sipas modelit *master/slave*. Në ndërkohë janë zhvilluar specifikacionet të bazuar në TCP/IP, UDP, Fieldbus, etj. Për këto specifikacione modeli i komunikimit është klient/server. Për ne me rëndësi është Modbus RTU i realizuar mbi komunikimin serial [44]. Disa nga karakteristikat janë si vijon. Ky protokoll është *master/slave*. Vetëm një *master* në të njëjtën kohë është i lidhur në bus të komunikimit me disa *slave* (maksimum 247). Vetëm *master*-i mund të inicojë kërkesë. *Slave*-t nuk mund të transmetojnë të dhëna pra marrë kërkesë nga *master*-i. *Slave*-t nuk komunikojnë me njëri tjetrin. Çdo kërkese të *master*-it mund të i përgjigjet vetëm një *slave*. *Masteri* në të njëjtën kohë mund të inicojë vetëm një transaksion. Kjo mundësohet përmes mesazheve të përcaktuar si në figurën 2.4. Adresa e *slave* fillon nga vlera 1 (deri në 247) për modën unicast, vlera 0 është e rezervuar për adresim në modën broadcast. Te moda unicast çdo kërkese të *master*-it i korespondon një përgjigje nga *slave* përkatës. Në modën broadcast të gjithë *slave*-t e pranojnë dhe ekzekutojnë kërkesën por asnjëri nuk përgjigjet me mesazh. Kuptimi i fushave tjera të mesazhit është evident. Madhësia maksimale e mesazhit mund të jetë 256 bajtë. Mesazhet dërgohen njëri pas tjetrit duke pauzuar në kohë ekuivalente prej minimum 3½ karakterëve.

Adresa e slave	Kodi i funskionit	Të dhënat	CRC	
1 bajt	1 bajt	0 deri në 255 bajt(ë)	2 bajtë	
			CRC Low	CRC Hi

Figura 2.4 Formati i Modbus RTU (serial) mesazhit (sipas [44])

Në tabelën 2.1 në vijim janë paraqitur të përmbledhura karakteristikat e protokollëve të posa analizuar: EXI, CoAP dhe Modbus RTU.

Tabela 2.1 Karakteristikat e protokolleve relevantë EXI, CoAP dhe Modbus RTU (* relativisht me njëri tjetrin)

Protokolli	**A. O. Sh.**	**Enk./Komp.**	**Komunik.**	**Madh. mes.**	**Madh. e kod.***	**I përshkrueshëm**
EXI	Po	Po	TCP	~ < 10 bajtë	Madhe	Po (WSDL)
CoAP	Po	Po	UDP	~ 10 bajtë	Mesatare	Po (WADL)
Modbus RTU	-	Jo	Serial	~ 10 bajtë	Vogël	Jo

Nga tabela 2.1 mund të përfundojmë se EXI dhe CoAP është Arkitekturë e Orientuar në Shërbime (A.O.Sh.), ndërsa Modbus RTU është i bazuar në logjikën *master/slave* (nuk është A.O.Sh.). A.O.Sh.-të janë treguar të përshtatshme (në krahasim me zgjidhjet ad-hoc si p.sh. *application wrapper*) për shkak të karakteristikave themelore, që janë pavarësia nga prodhuesi, produkti dhe teknologjia. Më tej, EXI dhe CoAP gjatë komunikimit aplikojnë komprimim/kodim në grupe bitësh, ndërsa Modbus RTU jo. Në rastin tonë, duhet përdorur protokoll të pakomprimuar për shkak të përpunimit më të lehtë, dhe rrjedhimisht kodit më të vogël rezultues. Për teknologji komunikimi EXI përdor TCP, CoAP përdor UDP, ndërsa Modbus RTU atë serial. TCP/UDP janë më të përshtatshme për shkak të shpejtësisë më të madhe (në krahasim me p.sh. Modbus RTU) prandaj mundësisht që pjesa më e madhe e sistemit të realizohet përmes tyre. Sa i përket madhësisë së mesazheve të gjitha sillen aty rreth ~ 10 bajtë, me theks në EXI që janë diçka më të vogla. Kjo rezulton në kod më të madh (relativisht me njëri tjetrin) për EXI, pasuar nga CoAP dhe në fund Modbus RTU. Ndërsa sa i përket përshkrimit, EXI dhe CoAP janë të përshkrueshme me gjuhët e dedikuara WSDL dhe WADL respektivisht, ndërsa Modbus RTU jo.

2.3 Disa anë negative të të arriturave

Në punimin [15] zgjidhja që është ofruar paraqet zgjidhje ad-hoc. Kjo do të thotë se për çdo problem të dhënë (numër dhe shpërndarje sensorësh) duhet të shkruhen manualisht kodet për Shërbimet Ueb përkatëse të nyjeve me funksionalitet të plotë dhe atyre të reduktuar. Pra, mungon qasja me vegla, përveç gjenerimit të shtyllave dhe skeleteve të Shërbimeve Ueb.

Van Engelen [14] prezanton rrethinën zhvilluese gSOAP, të specializuar për gjenerim të kodit dhe optimizime runtime për zhvillim në C dhe C++ të Shërbimeve Ueb XML të lehta për pajisje të mbyllura. Dobësi kryesore e këtij punimi është se për kodin e gjeneruar, edhe pse të optimizuar, përdoren protokollet e Shërbimeve Ueb të bazuar XML standard që janë të papërshtatshëm për të u implementuar në rrethina të kufizuara. Edhe madhësia e kujtesës e nevojshme për punë është e papërshtatshme. Dokumenti [45] raporton madhësi kodi të gjeneruar prej 73 kB deri në 100 kB, dhe

kujtesë të nevojshme për të dhëna prej 2 kB, që janë shumë më të mëdha se ato të mikrokontrollorit tonë prej 12 kB madhësi kodi dhe 256 bajtë kujtesë të dhënash.

Në punimet e Käbisch [40][41][42] edhe pse është involvuar EXI, prapë komunikimi nuk është Shërbim Ueb i standardizuar pasi që nuk mund ta përdorim klientin standard të gjeneruar nga WSDL përshkrimi i Shërbimit Ueb të implementuar në mikrokontrollor. Në të dyja anët (klient dhe server) është i nevojshëm Procesori EXI për kodim/dekodim të mesazheve të shkëmbyera.

Përderisa EXI [27] është i bazuar tërësisht në komprimim të mesazheve edhe CoAP [28] përdor kodim në nivel bitësh të të dhënave. Edhe pse kjo qasje, e sidomos te EXI, bën që madhësia e mesazheve të shkëmbyera dukshëm të zvogëlohet, kjo paraqet punë shtesë në përpunimin (dekomprimimin) e tyre. Kjo vlen sidomos për EXI-n. Edhe te CoAP paraqitet nevoja e manipulimit me bitë, si zhvendosja e tyre majtas ose djathtas për ta përgatitur informatën binare për dekodim për krahasim me vlerën e pritur. Rasti i njëjtë është edhe me ndërtimin e mesazhit (kodim).

Më i përshtatshëm në rastin tonë duket se është Modbus [43]. Në këtë rast alternativat kodohen në bajtë të veçantë. Kjo e bën të përshtatshëm për aplikim në pajisje të mbyllura 8 bitëshe, e sidomos në rastin e mikrokontrollorit 8051. Kjo për arsye të ekzistencës së instruksioneve të krahasimit që janë të orientuara në bajt (8 bite). Te ky protokoll mangësitë vijnë në shprehje kur kemi të bëjmë me më shumë alternativa. Kodimi i secilës alternativë në bajt të veçantë do ta rriste madhësinë e mesazheve të shkëmbyera. Duhet të theksohet se shumë alternativa do të paraqiteshin në rastin e kodimit të ndërfaqeve komplekse të funksionaliteteve që servisi ofron. Në rastin e ndërfaqeve të thjeshta, që në raste konkrete aplikimi janë më të shpeshta, Modbus ose protokoll i ngjashëm me të do të paraqiste zgjedhjen e duhur.

KAPITULLI 3

Propozimi për kontribut në tezë

3.1 Hyrje

Duke marrë parasysh punimet e prezantuara në kapitullin 2 mbi përdorimin e pajisjeve të mbyllura me qëllim marrje të dhënash dhe dirigjim, ne propozojmë kontributet vijuese në këtë punim teze për doktoraturë.

Kontributet kryesore të tezës janë:

Standardizimi i mënyrës së komunikimit me pajisjet primitive (me theks në mikrokontrollorin 8051 [3])

Arkitekturë e orientuar në servise e komunikimit, në mënyrë që pajisja primitive (mikrokontrollori) të jetë e qasshme përmes rrjetës lokale dhe Internetit

Arkitekturë back-end e pavarur nga platforma, ashtu që pajisjes primitive (mikrokontrollorit) njëkohësisht mund t'u qasemi nga platforma të ndryshme

Realizimi i A.O.Sh.-ve në pajisjet primitive me ndihmën e veglave të quajtura gjeneratorë kodi

3.2 Standardizimi i mënyrës së komunikimit me pajisjet primitive (me theks në mikrokontrollorin 8051 [3])

Me standardizim të mënyrës së komunikimit është menduar krijimi dhe aplikimi i një mënyre unike të shkëmbimit të dhënave me pajisje primitive. Rëndom kjo nënkupton projektimin e protokollit përkatës. Protokolli përcakton formatin e mesazhit për t'u shkëmbyer dhe rregullat për shkëmbimin e këtyre mesazheve në mes të pajisjeve të sistemit. Protokolli mund të jetë i shtresuar d.m.th. një protokoll mund të ndërtohet mbi ndonjë protokoll tjetër. Protokolli kompleks ndërtohet mbi protokoll më të thjeshtë, për shembull, protokolli që implikon shkëmbimin e disa bajtëve mund të ndërtohet mbi protokollin që bart një bajt të vetëm. Për më tepër, protokolli mund

të jetë binar ose tekstual. Protokolli binar ka përparësi mbi protokollin tekstual kur kemi të bëjmë me pajisje të mbyllura pasi që është më i lehtë për të u përpunuar në krahasim me "llafazanërinë" e protokollit tekstual. Protokolli binar rezulton me kod të vogël të përpunimit dhe kujtesë të vogël. Kohëve të fundit protokollet tekstuale janë rëndom të bazuar në XML. Si alternativë binare të formatit XML ne kemi parë EXI [27]. Tjetër protokoll binar për pajisje të limituara me theks në mikrokontrollorë kemi parë CoAP-in [28]. Të dy në princip komprimojnë mesazhet, që rezulton në kujtesë të vogël të nevojshme për të ruajtur mesazhin, por rrisin madhësinë e kodit të nevojshëm për ta dekomprimuar dhe përpunuar atë. Për të ilustruar, CoAP ruan në header disa flags dhe parametra brenda një bajti të vetëm duke e ndarë bajtin në grupe të bitëve ku secili grup ka domethënie të veçantë. Poashtu, kemi parë që në EXI secili bit paraqet vendim në kalim të gramatikës në vend të përdorimit të okteteve, që paraqet punë shtesë në nxjerrje të bitëve të veçantë nga okteti. Duke i pasur këto parasysh ne propozojmë protokoll që është kompromis ndërmjet madhësisë së mesazhit dhe kodit të nevojshëm për ta ndërtuar/përpunuar atë. Protokolli është shumë i thjeshtë, siç është treguar në figurën 3.1, dhe është quajtur Adaptive Minimal Inter Server Protocol (AMISP). Emri vjen nga fakti që fillimisht mikrokontrollori ishte konsideruar rreptësisht si server, por me pak punë, ai mund të përdoret edhe kur mikrokontrollori vepron si klient d.m.th. inicion operacione.

1 bajt	1 bajt	1 bajt	1 bajt	n bajt(ë)	m bajt(ë)	…
Gjat. e mes.	ID e ndërf.	ID e oper.	Statuti	Param. 1	Param. 2	…

Figura 3.1 Komponentet fragmentuese të AMISP mesazhit

Protokolli AMISP është i projektuar për familjen e mikrokontrollorëve 8051, për të u transferuar përmes portës seriale RS232 të integruar të komunikimit, por mund të përdoret edhe për pajisje të tjera. Bajti i parë i *stream*-it është gjatësia e *stream*-it, kështu gjatësia e mesazhit është e kufizuar në 256 bajtë. Kjo nuk paraqet problem pasi që në konfiguracionin e tij themelor, kur nuk është i aplikuar zgjerimi i kujtesës, mikrokontrollori 8051 ka vetëm 128 bajtë të lirë në RAM, që në të shumtën e rasteve janë të mjaftueshëm për të identifikuar çdo kërkesë që vjen në mikrokontrollor. Kjo bëhet edhe më bindëse kur siç do të shohim më vonë operacioni

për të u thirrur në mikrokontrollor kodohet në një bajt të vetëm në vend të *signature* të tij të plotë. Bajti i dytë është kodi identifikues i ndërfaqes. Ky bajt është i rëndësishëm kur më shumë ndërfaqe kanë për të u implementuar në mikrokontrollor, tjetër mikrokontrollor ka për t'u komunikuar përmes një porte të vetme, ose, kur mikrokontrollori shërben për më shumë qëllime dhe kodi duhet të ndahet fizikisht brenda kujtesës së kodit të mikrokontrollorit. Në praktikë rëndom paraqitet rasti i parë. Bajti i ndërfaqes (me ngjyrë të himtë të hapur) është opsional dhe nuk gjenerohet kur mikrokontrollori implementon vetëm një ndërfaqe. Kjo është e arsyeshme sepse nënkuptohet se është fjala për të vetmen ndërfaqe. Kur ka më shumë duhet të gjenerohet që të bëhet dallimi me të tjerët se cili është duke u adresuar. Bajti i tretë opsional është operacioni (rutina) që thirret në mikrokontrollor. Rëndom mikrokontrollori kryen disa operacione dhe ato identifikohen përmes kodit të tyre identifikues në vend të *signature* të tyre. Për shembull, vlera që duhet të lexohet nga sensori 1 identifikohet me kodin zero. Edhe ky bajt gjenerohet vetëm në rastin kur në ndërfaqe janë të deklaruar më shumë se një operacion, që të bëhet dallimi. Bajti i katërt (ngjyrë hiri të mbyllur) paraqet statutin e kryerjes së operacionit dhe figuron vetëm në përgjigjen që kthehet nga pajisja e mbyllur. Nëse operacioni është kryer me sukses ky bajt ka vlerën 0, nëse ka ndodhur gabim kthehet kodi i gabimit (më i madh se zero). Kodet e gabimeve caktohen ashtu që gabimet globale marrin vlera nga 1 deri në 127, ndërsa ato lokale nga 128 deri në 255. Gabimet globale, siç tregon edhe vetë emri kanë kode të njëjta në operacionet e të gjithë ndërfaqeve. Gabimet lokale edhe pse me kode të njëjtë, në ndërfaqe të ndryshme kanë kuptime të ndryshme – bëhet fjalë për gabime të ndryshme. Bajtët tjerë që vijnë janë parametrat që i përcillen operacionit (në kërkesë) dhe ato që i kthen operacioni (në përgjigje). Në kërkesë figurojnë parametrat hyrës dhe ata hyrës/dalës sipas rradhës që janë të deklaruar në operacion. Në përgjigje figurojnë parametri që kthen operacioni (si funksion jo-void), si dhe parametrat hyrës/dalës dhe ata dalës sipas rradhës që janë të deklaruar në operacion. Madhësia (numri i bajtëve) e parametrit varet nga tipi i tij. Kur kemi të bëjmë me shembullin paraprak me sensorin 1, parametrat hyrës mund të përmbajnë precizitetin e vlerës që do të lexohet nga sensori. Ngjashëm vlen edhe për bajtët që kthehen (përgjigjen) nga mikrokontrollori, ata mund ta përmbajnë vlerën e lexuar nga sensori. Kodi identifikues i ndërfaqes dhe kodi identifikues i rutinës në përgjigje janë

të njëjtë sikur në kërkesë. Mund të vërejmë që nuk ka bajt që identifikon nëse mesazhi është kërkesë ose përgjigje. Kjo nuk është e nevojshme pasi që mikrokontrollori mund të jetë, ose server sikur në të shumtën e rasteve, ose klient. Në këtë mënyrë, kur mikrokontrollori punon si server, është e qartë që mesazhet që i vijnë atij janë kërkesa ndërsa mesazhet që kthehen nga ai janë përgjigje. Ngjashëm kur punon si klient mesazhet që i dërgon janë kërkesa ndërsa ato që i pranon janë përgjigje. Duhet theksuar se në këtë punim mikrokontrollori trajtohet strikt si server. Tjetër kufizim është se komunikimi është sinkron d.m.th. vetëm një kërkesë pranohet, ekzekutohet dhe përgjigjet në të njëjtën kohë nga të gjithë mikrokontrollorët e lidhur në portë. Sinkronizimi në komunikim eliminon nevojën për identifikim të mesazhit (bajt identifikues) pasi që, për mesazhin e dhënë kërkesë pritet për mesazh përkatës përgjigje. Mund të konkludohet se AMISP është i përpunueshëm si seri e bajtëve individual, ku secili bajt paraqet një entitet, për dallim nga protokollet tjerë të përmendur që brenda bajtit bitet mund të kenë domethënie të ndryshme. Kjo është e përshtatshme për mikrokontrollorin (dhe pajisjet tjera primitive) pasi që instruksionet e tij veprojnë në një bajt (instruksione 8 bitëshe) të tërë ose bit të vetëm. Kështu mundësohet krahasimi i lehtë i vlerave të marra nga porta e komunikimit dhe atyre të pritura. Krahasimi i bitit të vetëm është i pa zbatueshëm, pasi që me të mund të kodohen vetëm dy opsione të ndryshme.

Të qenit protokoll binar është përparësi e AMISP pasi leximet nga sensorët dhe vlerat që pranojnë aktuatorët janë rëndom binare. Kështu nuk ka nevojë konvertimi nga vlera binare në atë tekstuale, që do të mund të kërkohej gjatë shkëmbimit të mesazheve te protokollet tekstuale siç është p.sh. SOAP (XML).

3.3 Arkitekturë e orientuar në servise e komunikimit, në mënyrë që pajisja primitive (mikrokontrollori) është e qasshme përmes rrjetës lokale dhe Internetit

Konsorciumi W3C [29] përkufizon arkitekturën e orientuar në shërbime (AOSh) si “Bashkësi të komponenteve që mund të thirren, dhe përshkrimet e ndërfaqes mund të publikohen dhe zbulohen”. Teknologjitë aktuale të orientuara në

shërbime rëndom implementojnë logjikën e sipërpërmendur. Ato janë të organizuara sipas modelit klient/server. Servisi dhe funksionalitetet që ai ofron janë të përshkruara plotësisht përmes gjuhës së teknologjisë përakatëse. Gjuha që përshkruan funksionalitetet tenton të jetë e pavarur nga gjuha programuese. Po ashtu, në çdo teknologji të caktuar, janë mekanizmat përkatës për të zbuluar servisin. Ne nuk synojmë të zhvillojmë teknologji të re dedikuar mikrokontrollorit 8051, në vend të kësaj, ne synojmë të integrojmë mikrokontrollorin me teknologjitë aktuale të AOSh të lëmisë. Përfitimet e një pune të tillë janë të numërta. Përfitimi kryesor është lidhur me rrjetëzimin. Me këtë rast, nuk do të kishim nevojë të zhvillonim mekanizma lidhur me p.sh. hostimin e servisit ose transportimin e kërkesave/përgjigjeve nëpër rrjet, të cilat do t'i merrnim të gatshme nga teknologjitë A.O.Sh.. Ne tanimë i kemi përmendur shkurtimisht përparësitë dhe mangësitë e zhvillimit të teknologjisë së re dhe përdorimit të ekzistueseve. Po i përmendim ato përsëri. Zhvillimi i teknologjisë së re nga fillimi është shpenzim në kohë dhe burime. Do të ishte punë vërtetë e vështirë, duke pasur parasysh se zhvillimi i teknologjisë së re do të kërkonte paraprakisht një studim të thellë mbi përparësitë dhe mangësitë e teknologjive ekzistuese të lëmisë dhe marrjen në konsideratë të tyre gjatë projektimit dhe implementimit të teknologjissë së re. Jo-kompatibiliteti me teknologjitë ekzistuese dhe nevoja për trajnim të njerëzve për ta përdorur atë është një tjetër mangësi. Përparësi do të mund të ishte performansa për shkak të reduktimit të trafikut në pajisje të limituara. Qasje e dobishme do të ishte bërja e mikrokontrollorëve të përputhshëm me teknologjitë aktuale të popullarizuara siç janë Shërbimet Ueb [22], CORBA [23], dhe .NET Remoting [24]. Është e qartë se për shkak të kompleksitetit të tyre ato nuk mund të implementohen në rrethina të kufizuara. Mikrokontrollori do të mund të ekzekutonte versionin e thjeshtuar të teknologjisë përkatëse. Më tej, është e nevojshme të ndërtohet ndërfaqe ose adapter që do të ndërmjetësonte në mes të versionit të plotë dhe atij të thjeshtuar të teknologjisë përkatëse. Punimi [15] ka të bëjë me këtë problem. Në [15] pajisja e mbyllur është implementuar si REST servis, përderisa aplikacioni i shfrytëzuesit i qaset asaj si SOAP servis standard. Logjikisht, ndërfaqja është implementuar në nyjën e plotë funksionale (PC) që është i lidhur me pajisjen e limituar. Ne ndjekim logjikë të ngjashme, por ne nuk kufizohemi në teknologji të veçantë, por siç do të shohim në paragrafin vijues, mikrokontrollorit do të mund të i qasemi në të njëjtën kohë nga

teknologji të ndryshme. Kjo është e mundur duke implementuar protokoll të pavarur nga platforma sikur AMISP të paraqitur në paragrafin paraprak.

3.4 Arkitekturë back-end e pavarur nga platforma, në mënyrë që pajisjes primitive (mikrokontrollorit) njëkohësisht mund t'u qasemi nga platforma të ndryshme

Gjatë projektimit të protokollit AMISP ne kemi pasur në mendje thjeshtësinë dhe jo-varëshmërinë nga cilado teknologji e veçantë. Në vështrim të parë mund të vërehet se rutinat e klasës/programit janë të realizueshme në protokoll. Elemente tjera me rëndësi që duhet të përkrahë protokolli AMISP janë atributet. Me një mjeshtri të vogël ato mund të implemetohen lehtë me AMISP. Atributet mund te lexohen dhe vendosen. Secili nga këto veprime mund të konsiderohet si operacion standard. Këto janë konstruksione themelore të çdo klase/programi. Në të vërtetë, mikrokontrollori nuk është i aftë dhe në shumë raste praktike nuk është e nevojshme të realizohen konstruksione komplekse, siç janë për shembull listat me operacionet përkatëse. Për mikrokontrollorin është e mjaftueshme që ti komandojë sensorët për të lexuar vlerë nga ata ose të furnizojë aktuatorët me vlerë komanduese. Veprimet më të rëndomta janë transformimi i bajtëve në bitë dhe vendosja e këtyre bitëve në pinë dhe anasjelltas leximi i bitëve nga pinët dhe transformimi i tyre në bajtë. Të dyja këto veprime mund të jenë të pranishme gjatë një operacioni të vetëm.

Pasi që konstruksionet bazike të klasës/programit janë të realizueshme në teknologjitë e rëndomta AOSh të përmendura, ne kemi ardhur në idenë që mikrokontrollorit mund të i qasemi nga secila prej tyre. Për më tepër, ne kemi bërë pyetjen nëse atij mund të i qasemi në të njëjtën kohë nga secila prej tyre. Të qasurit mikrokontrollorit në të njëjtën kohë është problem i menaxhimit të lidhjeve. Në të vertetë, është e pamundur t'u qasemi në të njëjtën kohë, kështu që serveri lidhës përkatës lidhet me mikrokontrollorin, dërgon kërkesën, merr përgjigjen dhe pastaj shkëputet nga ai për të ia liruar lidhjen serverit tjetër lidhës. Serveri tjetër lidhës mund të jetë i teknologjisë së njëjtë me të parin (kur përkrahen më shumë ndërfaqe) ose i një

teknologjie tjetër. Të gjithë këta duhet të komunikojnë mikrokontrollorin përmes protokollit AMISP siç është paraqitur në figurën 3.2.

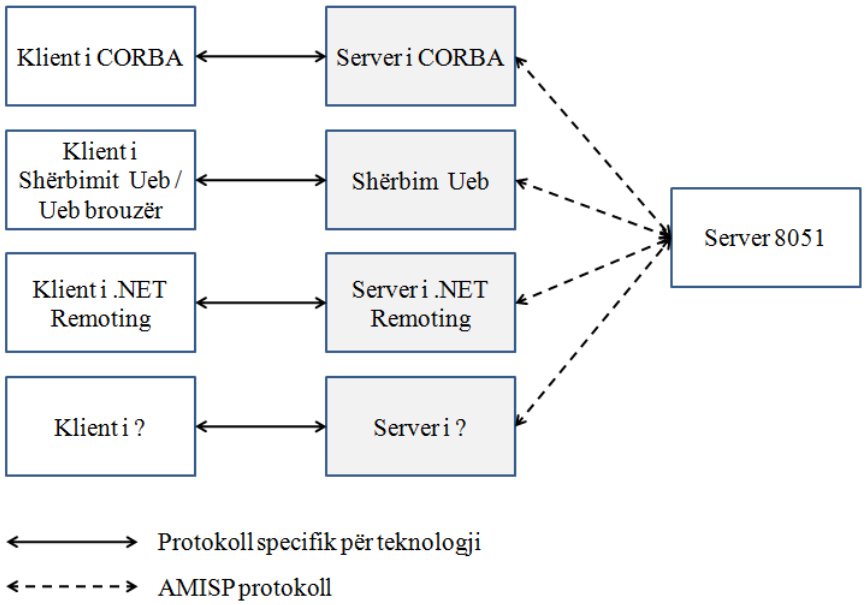


Figura 3.2 Përkrahja për shumë teknologji për mikrokontrollorin 8051

Serveri i ndërmjetëm quhet po ashtu edhe server lidhës. Figura 3.2 tregon qajsen e serverit 8051 nga klienti përmes serverit të ndërmjetëm (lidhës) i treguar në kuti të përhimëta, për disa teknologji të rëndomta. Klienti dhe serveri i ndërmjetëm komunikojnë përmes protokollit përkatës specifik të standardizuar për teknologjinë e veçantë. P.sh. për rastin e CORBA-s klienti dhe serveri i ndërmjetëm do të komunikonin përmes protokollit IIOP (ang. Internet Inter Orb Protocol). Në rastin e Shërbimeve Ueb do të komunikonin përmes protokollit SOAP (ang. Simple Object Access Protocol). Pastaj serveri i ndërmjetëm dhe serveri 8051 komunikojnë përmes AMISP. Ai i përkthen mesazhet nga protokolli specifik për teknologjinë (CORBA, Shërbime Ueb) në AMISP të "kuptuar" nga mikrokontrollori. Ai është specifik për atë teknologji. Në të vërtetë, qasja jonë është e orientuar nga gjuha programuese, dhe me disa modifikime rreshtash kodi i serverit të ndërmjetëm mund të përdoret në teknologjitë tjera me kushtin që teknologjia të përkrahë zhvillimin e serverëve në gjuhën e caktuar programuese. Kjo do të thotë që kodi i serverit lidhës të zhvilluar në Java për CORBA (JacORB) me disa modifikime rreshtash mund të përdoret për serverin lidhës në Java për alternativën e Shërbimeve Ueb.

Mund të mendohet se një aplikacion *wrapper* mund ta kryente rolin e serverit lidhës. Aplikacioni *wrapper* do të ishte jo-standard dhe para së gjithash i mangët. Jo-standard sepse do të duhej të përkufizohej gjuha për përcaktim dhe qasje të tij, siç është p.sh. IDL në CORBA dhe WSDL te Shërbimet Ueb. I mangët pasi që në vend se të shfrytëzoheshin funksionalitetet e AOSh aktuale, ne do të duhej të zhvillonim mekanizma p.sh. për menaxhim të lidhjeve dhe shkëmbim të përshtatshëm të mesazheve me serverin në pajisje të mbyllur përmes aplikacionit *wrapper*. Me fjalë të tjera do të shpenzonim kohë në ri-projektim të teknologjive të reja në vend se të shfrytëzoheshin karateristikat dhe përparësitë e atyre ekzistuese të lëmisë.

Përparësi e serverit lidhës është se ai bën ndarjen e përgjegjësive në mes tij dhe serverit të realizuar në pajisje të mbyllur të kufizuar. Ai merr përsipër punët komplekse siç është qasja nga më shumë klientë (shkëmbimi i mesazheve me klientin), menaxhimi i lidhjeve, mundësia e lidhjes të më shumë serverëve të mbyllur në të njëjtën portë dhe adoptimin e mesazheve nga një protokoll në tjetrin. Çka i mbetet serverit në pajisje të mbyllur është të merret me komunikimin me sensorë dhe aktuatorë dhe komunikimi i vlerave në mënyrë të thjeshtuar me serverin lidhës. Serveri lidhës merret me punë komplekse të përmendura kurse serveri në pajsije të mbyllur me ato të thjeshta.

3.5 Realizimi i A.O.Sh.-ve në pajisjet primitive me ndihmën e veglave të quajtura gjeneratorë kodi

Zgjidhja e orientuar në servise e propozuar nën b) (paragrafi 3.3) skematikisht është paraqitur nën c) (paragrafi 3.4). Fillimisht, serveri i ndërmjetëm dhe serveri 8051 janë zhvilluar manualisht. Kjo qasje (manuale) i është nënshtruar gabimeve, shpenzon kohë dhe kërkon trajnim të lartë të njerëzve të angazhuar. Duke i marrë parasysh këto ne kemi shikuar për mundësinë që ta bëjmë atë automatikisht d.m.th. të gjenerojmë kodin e nevojshëm me ndihmën e veglave. Ne e kemi optimizuar kodin manualisht dhe kemi bërë përgjithësime aq sa ka qenë e mundur. Ne kemi bazuar zhvillimin tonë në JacORB [30], implementim në Java i CORBA-s [23]. Në vijim, po e përshkruajmë punën nga aspekti i qasjes.

Fillimisht, kemi punuar në serverin 8051. Çka duhet të gjenerohet në serverin 8051 është komunikimi me serverët e ndërmjetëm. Implementimi i operacioneve duhet të shkruhet manualisht nga zhvilluesi sepse ato janë specifike për aplikacion. Ne kemi punuar vetëm në gjenerim të kodit për komunikim. Në anën tjetër serverët e ndërmjetëm janë gjeneruar tërësisht. Dihet që në CORBA [23] serveri përshkruhet në skedarin IDL (Interface Definition Language). IDL është e pavarur nga gjuha programuese dhe është e dedikuar për të përshkruar funksionalitetet që serveri ofron. Sintaksa e IDL është e përcaktuar me gramatikë të atribuuar dhe specifikacioni i saj i versionit 3.3 [31] përmban gjithsej 138 rregulla. Është e qartë, kjo gramatikë nuk mund dhe në raste praktike nuk është e nevojshme që të implementohet tërësisht në mikrokotrollorin 8051 [3]. Ne kemi ekstraktuar ato nënbashkësi nga gramatika [31] që realizojnë ndërfaqet me operacionet standarde, operacionet njëkahëshe (oneway) si dhe atributet (ata të rëndomtë dhe vetëm të lexueshëm). Poashtu, gramatika përkrahë edhe gabimet (ang. exceptions), që sinjalizohen nga operacionet standarde. Gabimet mund të jenë lokalë – që vlejnë vetëm brenda ndërfaqes ku janë deklaruar, si dhe globalë – që vlejnë për të gjitha ndërfaqet brenda skedarit IDL. Pastaj, gramatikën e kemi transformuar në mënyrë që të jetë e pranueshme nga vegla që gjeneron kompilatorë Coco/R [1]. Ato rregulla gramatikore janë paraqitur në listingun 3.1 në vijim.

```
 1 AMISPIDL = [ "#include" "\"ISEASecured.idl\"" ] { interfacedef |
exceptiondef }.
 2 interfacedef = "interface" identifier [":" "ISEASecured" ] "{"
interfacebody "}" ";".
 3 interfacebody = {interfacelement}.
 4 interfacelement = methoddef | attributedef | exceptiondef.
 5 methoddef = onewaymethod | standardmethod.
 6 onewaymethod = "oneway" "void" identifier "("
[onewaymethodparammetersbody] ")" ";".
 7 standardmethod = simpletypesplusvoid  identifier "("
[methodparammetersbody] ")" [ "raises" "(" identifier { ","
identifier } ")" ] ";".
 8 simpletypes = "octet" | "char" | "boolean" | "short" | "float" |
"double" | "long" ["long" | "double"] | "unsigned" ( "short" | "long"
["long"]).
 9 simpletypesplusvoid = simpletypes | "void".
10 onewaymethodparammetersbody = onewaymethodparammeter { ","
onewaymethodparammeter}.
11 methodparammetersbody = methodparammeter { "," methodparammeter}.
12 onewaymethodparammeter = "in" simpletypes identifier.
13 methodparammeter = ("in" | "out" | "inout") simpletypes
identifier.
```

```
14 attributedef = ["readonly"] "attribute" simpletypes identifier {
"," identifier } ";".
15 exceptiondef = "exception" identifier "{" { exceptionparam } "}"
";".
16 exceptionparam = simpletypes identifier { "," identifier } ";".
```

Listingu 3.1 Rregulla gramatikore të reduktuara të pranueshme nga Coco/R për të përcaktuar IDL

Mund të shihet që sintaksa e reduktuar e IDL është përshkruar me vetëm 16 rregulla gramatikore. Për ta kuptuar domethënien e rregullave referohuni kapitullit 4.2 ose literaturës [1]. Konfliktet LL(1) (d.m.th. e parsueshme prej majtas në të djathtë me derivime të majta kanonike dhe një simbol të vëzhgueshëm) që paraqiten në rregullën 8 dhe 13 janë zgjidhur siç është propozuar në [32]. Më tej, parametrat hyrës, dalës dhe i kthyer si dhe tipat e atributeve mund të jenë të tipave themelorë (të thjeshtë) që janë të madhësisë së fiksuar. Siç u përmend më parë, as stringje, as vargje, e as struktura tjera komplekse nuk janë përkrahur.

3.6 Propozimi për kontribut skematikisht

Kontributi i këtij disertacioni është paraqitur skematikisht në figurën 3.3.

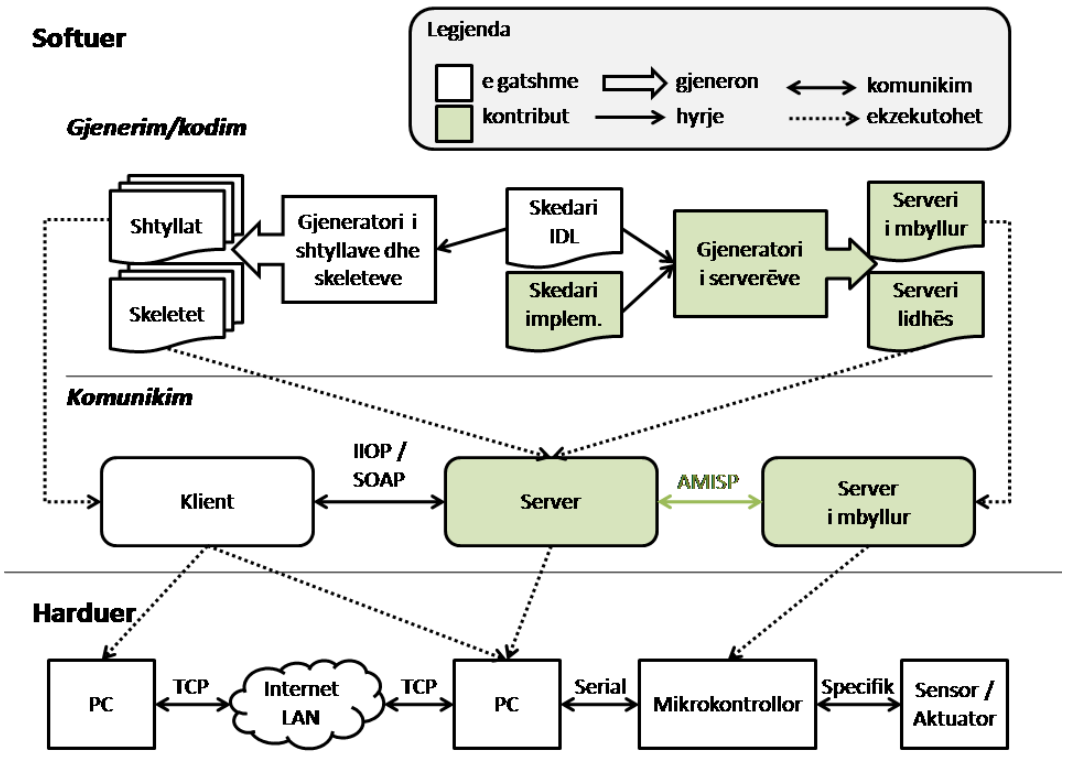


Figura 3.3 Skema e kontributit ndarë në shtresa

Në figurën 3.3 janë paraqitur shtresat e softuerit dhe harduerit dhe lidhja e tyre me kontributin. Specifikisht kontributi i disertacionit është paraqitur me ngjyrë të gjelbër dhe ai i përket shtresës softuerike.

KAPITULLI 4

Teknologjitë e involvuara në hulumtim

4.1 Hyrje

Në këtë punim do të shtjellohet integrimi i mikrokontrollorit 8051 (si përfaqësues tipik i pajisjeve shumë të kufizuara në burime dhe fuqi) në arkitektura të orientuara në shërbime përmes gjeneratorëve të kodit (burim). Prandaj do të ishte e logjikshme të njoftohemi dhe të japim përshkrimet bazë të teknologjive të përfshira në hulumtim. Në kapitujt në vijim do të prezantohet mikrokontrollori 8051, vegla për ndërtim të gjeneratorit të kodit Coco/R si dhe teknologjia e orientuar në shërbime CORBA me implementimin e saj në Java – JacORB.

4.2 Mikrokontrollori

Mikrokontrollori paraqet kompjuterin në një çip të vetëm. Në një çip të vetëm mund të gjendet Njësia Qendrore Procesorike (ang. CPU – Central Processor Unit), kujtesa e të dhënave (ang. RAM – Random Access Memory), kujtesa flesh për programim (ang. ROM – Read Only Memory) dhe kujtesa me veti të kombinuara RAM dhe ROM (ang. EEPROM – Electrically Erasable Programmable Read Only Memory). Sasitë e komponenteve të lartpërmendura janë të pakrahasueshme me ato që gjenden në një kompjuter personal të rëndomtë. Ato janë mjaftë të vogla që e bëjnë mikrokontrollorin të aplikueshëm për kryerjen e vetëm një pune. Ata rëndom gjenden në furra mikrovalë, automobilë etj. Poashtu konfigurimi i pinëve të tyre dhe mundësia e bit-adresimit i bëjnë të përshtatshëm për lexim të sensorëve dhe ngasje të aktuatorëve. Kështu ata janë të përshtatshëm për sisteme të marrjes së të dhënave dhe kontrollit. Dallohen nga mikroprocesorët në atë se instruksionet e tyre nuk janë intensive në të dhëna, duke qenë kështu rëndom 8-bitëshe. Poashtu instruksionet janë të orientuara në bite duke lejuar adresimin dhe qasjen e tyre në mënyrë të veçantë.

4.2.1 Standardi 8051

MCS-51 [3] është familje e mikrokontrollorëve fillimisht e zhvilluar dhe prodhuar nga kompania Intel Corporation. Pa hyrë në detaje historike do ta prezantojmë llojin e mikrokontrollorit 8051 (kompatibil me të tjerët) mbi të cilin është zhvilluar pjesa realizuese-praktike e këtij disertacioni. Ai është mikrokontrollori 8051 i llojit AT89S8253 i përshkruar me specifikacionin [5]. Do të i paraqesim vetëm karakteristikat themelore dhe ato që hyjnë në punë/gjenerohen nga gjeneratori i kodit.

4.2.2 Arkitektura

Arkitektura përbën komponentet në kuadër të çipit AT89S8253 si dhe përdorimin e tyre. Në kuadër të këtij kapitulli do të paraqiten përshkrimi i disa nga pinëve të çipit, organizimi i kujtesës, disa nga regjistrat me funksion special, interaptët, tajmerët si dhe komunikimi serial.

Karakteristikat kryesore të AT89S8253 janë:

Kompatibil me familjen 8051.

12 kilobajtë të kujtesës flesh për ruajtje të programeve.

2 kilobajtë të kujtesës EEPROM.

Tension i furnizimt prej 4 V deri në 6 V.

Frekuencë e punës prej 0 MHz deri në 24 MHz.

256 bajtë të RAM të brendshëm për ruajtje të variablave.

32 pinë hyrës/dalës.

Tre tajmerë/numërues 16 bitësh.

9 burime interapti.

Komunikim serial të programueshëm.

4.2.3 Përshkrimi i pinëve

Lloji i çipit që ne përdorim është PDIP 40 pinësh i paraqitur në figurën 4.1 në vijim:

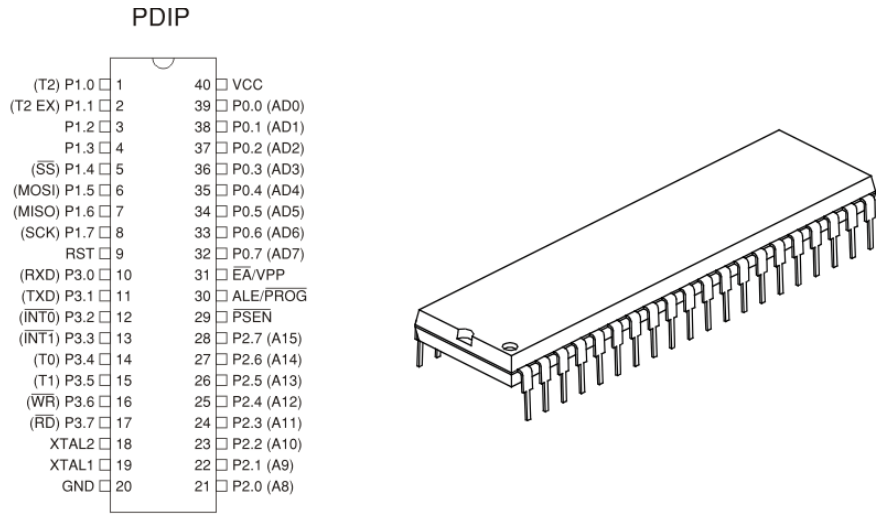


Figura 4.1 Pinët e çipit (majtas) dhe dukja fizike (djathtas) (sipas [4])

Domethënia e pinëvë është evidente, P0.1 deri në P0.7 janë pinët e portës 0, njëlloj janë edhe për pinët e portave të tjera (P1.x, P2.x dhe P3.x). Në kuadër të portave ka pinë me dedikim të posaçëm siç janë p.sh. P3.0 (RXD) pini i cili shërben për pranim të dhënash përmes portës seriale të integruar të çipit; P3.1 (TXD) pini i cili shërben për transmetim të dhënash përmes portës seriale të integruar të çipit, etj. Pini VCC shërben për ta lidhur çipin për tension të vazhdueshëm të furnizimit (vlerat e lejuara prej 4 V deri në 6 V). GND është pini i tokëzimit.

4.2.4 Organizimi i kujtesës

Dallojmë kujtesat për program (ROM), RAM dhe EEPROM.

4.2.4.1 Kujtesa për program (ROM)

Kujtesa për program (ROM) ka madhësine prej 12 kilobajtëve dhe është realizuar në teknologjinë FLASH, që u mundëson programeve shumëherë të shkruhen në të dhe fshihen. Programohet përmes modulit të mbyllur SPI (ang. Serial Peripheral Interface). Nëse 12 kilobajtë nuk mjaftojnë, mund të shtohet edhe ROM çip i jashtëm.

4.2.4.2 Kujtesa e të dhënave (RAM)

Kujtesa RAM përbëhet prej 3 blloqeve që përmbajnë secili nga 128 regjistra (bajtë). Kjo strukturë bën pjesë në standardin 8051:

128 regjistra me aplikim gjeneral.

128 vende të kujtesës të rezervuara për regjistra me funksion të veçantë (ang. SFR – Special Function Register(s)). Edhe pse në të vërtetë vetëm disa vende përdoren si SFR, vendet tjerë të lirë nuk do të duhej të përdoren për ruajtje të variablave pasi janë të rezervuar për përdorim në të ardhmen.

128 regjistra shtesë në dispozicion, që nuk kanë ndonjë qëllim të veçantë. Pasi që këta regjistra i kanë adresat e njëjta me SFR, këtyre i qasemi përmes adresimit indirekt.

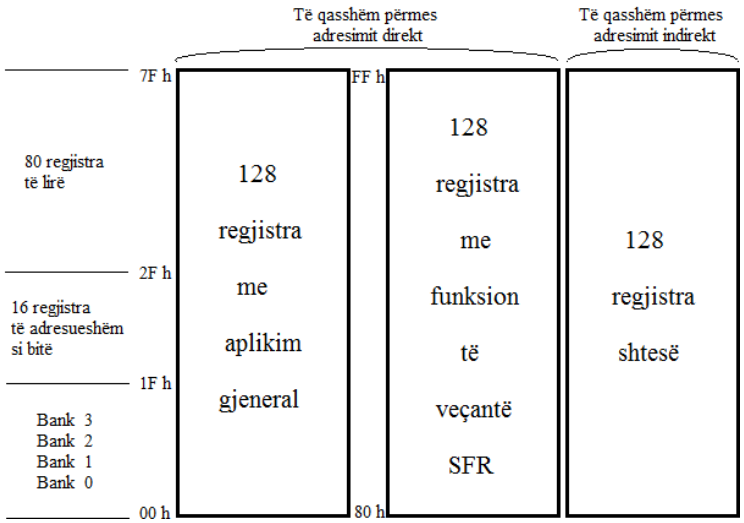


Figura 4.2 Organizimi kujtesës RAM në 8051 (AT89S8253)

4.2.4.3 Kujtesa EEPROM

EEPROM është lloj i veçantë i kujtesës që ka veti të kujtesave RAM dhe ROM. Përmbajtja e EEPROM mund të ndryshohet gjatë punës, por mbetet e ruajtur edhe pas shkyçjes nga furnizimi me tension. Mikrokontrollori AT89S8253 posedon në total 2 kilobajtë të kësaj kujtese.

4.2.4.4 Zgjerimi i kujtesës

Në rast se kujtesa e integruar (RAM dhe ROM) nuk mjafton, atëherë është e mundur të shtohen kujtesa çipë nga jashtë, secili nga 64 kilobajtë. Portat hyrëse/dalëse P2 dhe P3 përdoren për adresim të tyre dhe transmetim të dhënash.

4.2.4.5 Mënyrat e adresimit

Ekzistojnë dy mënyra të adresimit të kujtesës, direkt dhe indirekt, varësisht nga ajo se në operand të instruksionit specifikohet në mënyrë të drejtpërdrejtë regjistri i kujtesës apo përmes një regjistri tjetër.

4.2.5 Regjistrat me funksion të veçantë

Regjistrat me funksion të veçantë (ang. SFR) janë vendosur në kujesën RAM të mikrokontrollorit që shërbejnë për të ekzekutuar dhe monitoruar punën e mikrokontrollorit. Secili nga këta regjistra dhe secili bit në kuadër të tyre, e ka emrin e vet dhe adresën në RAM. Ata saktësisht përcaktojnë aplikimin e tyre siç është kontrolli i tajmerit, interaptit, portës seriale etj. Derisa në përgjithësi për mikrokontrollorët 8051 përcaktohen 21 regjistra të tillë, nga 128 vende bajtësh në total në dispozicion, për tipin AT89S8253 përcaktohen 40 regjistra SFR. Vendet tjera të pashfrytëzuara janë të rezervuara për përcaktime për tipa më të ri të mikrokontrollorëve, dhe këto vende nuk preferohet të përdoren.

4.2.6 Tajmerët

Tajmerët paraqesin lidhje seriale të flip-flopëve [3] që kanë si hyrje sinjalin nga oshilatori i kuarcit. Ky sinjal ka frekuencë konstante dhe mjaftë të qëndrueshme. Kështu sinjali vepron në flip-flopin e parë, dalja e të cilit vepron në flip-flopin e dytë, dhe kështu me rradhë. Në këtë menyrë flip-flopët paraqesin numratorë. Nëse tajmeri është 16 bitësh, do të thotë se mund të numrojë nga 0 (00000000000000002) deri në 216-1 = 65535 (11111111111111112). Pas vlerës 11111111111111112 kalohet në atë 00000000000000002, me ç'rast bëhet tejkalim dhe bëhet 1 biti i veçantë TFx (x – paraqet tajmerin, e që mund të jetë 0, 1 ose 2) që quhet bit i tejkalimit (ang. overflow).

4.2.7 Komunikimi (porta) serial

Mikrokontrollori 8051 e ka të integruar portën seriale ose UART (ang. Universal Asynchronous Receiver Transmitter) që i mundëson komunikim me pajisjet e jashtme, siç është për shembull kompjuteri personal. Kjo portë ka vetinë *full duplex* që do të thotë se në të njëjtën kohë mund të transmetojë dhe të pranojë karakter. Transmetimi bëhet përmes pinit TXD (P3.1) kurse pranimi përmes RXD (P3.0).

Tjetër veti që e karakterizon është *receive buffering*, që mundëson leximin e një bajti dhe mbajtjen e tij në bafer derisa të lexohet tjetri.

Porta seriale është shumë i thjeshtë për të u përdorur. Së pari ai duhet të konfigurohet, gjë që bëhet përmes bitëve të regjistrit SCON. Konfigurimi nënkupton sa bitë do të shkëmbehen dhe sa është frekuenca e shkëmbimit të bitëve. Leximi dhe transmetimi i bajtit bëhet përmes regjistrit SBUF. Në përdorim hyjnë edhe dy bitë (RI dhe TI) të regjistrit SCON që vendoset nga hardueri dhe tregojnë nëse bajti është pranuar ose transmetuar. Frekuenca e shkëmbimit varet nga moda e punës. Ajo mund të jetë e fiksuar që varet drejtpërdrejt nga frekuenca e oshilatorit të brendshëm ose mund të jetë variabile që varet nga vlerat e parametrave në tajmerin 1. Ne do ta përdorim modën 1 variabile të frekuencës.

4.2.8 Interaptët

Interapti është një ngjarje që shkakton ndërprerjen e përkohshme të ekzekutimit të programit, ekzekutim të një pjese të caktuar kodi dhe pastaj kthim në atë adresë ku kishte mbetur programi për t'u ekzekutuar. Është i ngjashëm me nënprogramin, por dallon nga ai sepse nuk dihet paraprakisht kur mund të ndodhë. Standardi 8051 përmban pesë burime standarde interaptësh, kurse lloji AT89S8253 përmban gjashtë burime. Ata janë: dy të jashtëm (që ndodhin me ndryshim gjendjeje në pinë ta caktuar), nga një për të tre tajmerët (T0, T1 dhe T2) dhe një nga porta seriale ose SPI (ang. Serial Peripheral Interface).

Interaptët mund të përpunohen ose jo varësisht nga bitët e vendosur në regjistrin e ndërprerjeve (IE). Nëse dy ose më shumë ndodhin në të njëjtën kohë, atëherë rradha e përpunimit të tyre bëhet sipas rradhës së paracaktuar. Nëse dëshirojmë që këtë rradhë ta ndryshojmë që t'ia përshtatim nevojave, atëhere kjo bëhet duke i caktuar vlerat e bitëve në regjistrin përkatës (IP).

4.2.9 Programimi

Procedura e programimit të mikrokontrollorit është relativisht e thjeshtë. Programi duhet të shkruhet në asembler ose gjuhë burimore si C, të përkthehet me kompilatorin përkatës dhe pastaj të shkruhet në mikrokontrollor përmes ndërfaqes së

integruar. Gjeneratorët tanë të kodit gjenerojnë kod në assembler për mikrokontrollorin 8051. Ky kod përmes kompilatorit C51ASM [26], përkthehet në format të përshtatshëm për shkruarje në mikrokontrollor. Ky format është i njohur me emrin IHEX (ang. Intel Hexadecimal), dhe është i organizuar sipas një strukture të caktuar që mundëson shkruarjen në mikrokontrollor përmes aplikacionit përkatës. Kujtesa ku regjistrohet programi në mikrokontrollor është ajo ROM flesh.

Programimi në asembler logjikisht mund të ndahet në dy kategori:

Instruksionet e mikrokontrollorit 8051, dhe

Elemente të deklarimeve asambler

4.2.9.1 Instruksionet e mikrokontrollorit 8051

Përmes këtyre instruksioneve ndërtohet logjika e punës së mikrokontrollorit. Logjika nënkupton përpunimin e ngjarjeve, transferin e të dhënave, vendimet e caktuara, etj. Pasi që mikokontrollori 8051 është i bazuar në arkitekturë 8 bitëshe, atëherë ai mund të përmbajë në total 255 instruksione, duke e rezervuar një instruksion për pa-instruksion (ang. NOP – no operand). Disa instruksione janë të ndryshme edhe pse nga sintaksa duken të njëjta/ngjashme. P.sh. INC R0, INC R1, ..., INC R7 paraqesin tetë instruksione të ndryshme, edhe pse ato llogariten si një duke e shënuar INC Rn. Në përgjithësi instruksionet mund të ndahen në këto grupe:

Instruksione aritmetike

Instruksione të degëzimit

Instruksione të transferit të të dhënave

Instruksione logjike

4.2.9.2 Elemente të deklarimeve asembler

Elementet e deklarimeve asembler përbëjnë konstruksionet të nevojshme për ndërtim të programit në gjuhën asembler. Gjuha asembler si çdo gjuhë programuese ka rregullat e veta. Këto rregulla për gjuhën asembler janë të thjeshta, ndër to dallojmë:

Komentet

Direktivat e asemblerit

Komentet e rrisin lexueshmërinë dhe kuptimin e kodit. Ato injorohen nga kompilatori i asemblerit, që do të thotë nuk përkthehen në asnjë instruksion të makinës. Komenti është tekst që shënohet pas shenjës së ; (pikëpresë). Pra, pjesa e instruksionit prej ; e më tej injorohet nga kompilatori i asemblerit.

Direktivat e asemblerit janë instruksione të programit në asembler që përcaktojnë strukturën e programit, të dhënat, konstantat etj. P.sh. direktivë është instruksioni *ORG Adresa* që tregon adresën në kujtesën programuese ku duhet të shkruhet kodi që vijon pas atij instruksioni.

4.2.10 Skedari HEX

Skedari HEX është skedari që fitohet pas kompilimit të skedarëve asembler. Ai përbëhet prej formatit të paraqitur në figurën 4.3. Aty shënohen të dhëna, që tregojnë se cila pjesë ku do të shkruhet në kujtesën programuese të mikrokontrollorit. Çdo rresht fillon me simbolin : (dy pika), pasohet me gjatësinë e rekordit (8 bitësh ose 2 shifra heksadecimale). Pastaj vjen adresa (16 bitëshe ose 4 shifra heksadecimale) në kujtesën programuese ku duhet të shkruhet rekordi që vijon, më tej vjen lloji i rekordit (8 bitësh ose 2 shifra heksadecimale). Në vijim vijnë të dhënat (rekordi) e gjatësisë së përcaktuar paraprakisht, dhe në fund vie shuma verifikuese (ang. checksum) (8 bitëshe ose 2 shifra heksadecimale) e të gjithë shifrave paraprake të rreshtit.

Shenja e rekordit	Gjatësia e rekordit	Adresa e ruajtjes	Lloji i rekordit	Të dhënat	Shuma verifikuese
:	0B	00A0	00	80FA92006F3600C3A000076	CB

Figura 4.3 Shembull i përmbajtjes së një rreshti të skedarit HEX

4.3 Vegla kompilatorike Coco/R

Coco/R [1] është vegël softuerike për ndërtim të gjeneratorëve të kodit, respektivisht kompilatorëve. Ajo merr si hyrje gramatikën e atribuuar të gjuhës burimore dhe gjeneron skanerin dhe parserin për atë gjuhë. Skaneri punon si automatë e fundme deterministike. Parseri përdor origjinën rekursive (ang. recursive descent) për analizë gramatikore. Konfliktet *LL(1)* mund të zgjidhen duke vëzhguar më shumë simbole vijuese ose duke u bazuar në kontrollë semantike. Kështu klasa e gramatikave të pranuara është *LL(k)* për një k arbitrare.

Ekzistojnë versione të Coco/R për gjuhët si C#, Java, C++, Delphi dhe të tjera. Por ne do ta përdorim versionin në Java, gjuhë në të cilën janë zhvilluar edhe gjeneratorët tanë të kodit.

4.3.1 Pamje e përgjithësuar

Siç u përmend në paragrafin 4.3 në bazë të gramatikës së gjuhës burimore Coco/R gjeneron skanerin dhe parserin. Çka i mbetet përdoruesit është të krijojë klasën kryesore (ang. main class) që thërret parserin, si dhe të i krijojë klasat semantike që mund të jenë tabela simbolike ose gjeneratori i kodit, figura 4.4.

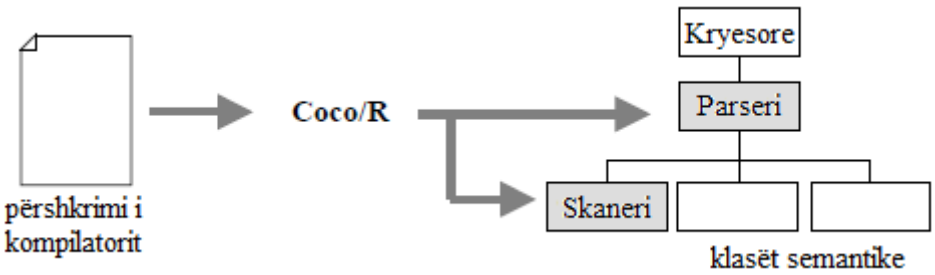


Figura 4.4 Hyrja dhe dalja e Coco/R (sipas [1])

Rëndom klasat semantike perdoren nga aksionet semantike në parser, por, për arsye të ndryshme mund të ekzistojnë edhe qasje tjera siç është ajo e jona, ku fillimisht ndërtohet lista e objekteve (e ngjashme me pemën sintaksore) dhe pastaj nga kjo listë bëhet gjenerimi i kodit. Arsyeja për këtë qasje bazohet në atë se për gjenerim

korrekt dhe të optimizuar të kodit nevojiten më shumë se një përshkim i strukturës së hyrjeve (skedarëve IDL dhe implementues) e që praktikisht ruhen në listë të objekteve.

Aksioni semantik është pjesë e kodit që shkruhet në gjuhën që përdoret në Coco/R (në rastin tonë Java) dhe ekzekutohet nga parseri i gjeneruar në pozitën e tij në prodhim. Atributet specifikojnë parametrat e simboleve. Ka atribute hyrëse dhe dalëse.

4.3.2 Gjuha hyrëse

Gjuha përshkruese e kompilatorit Coco/R shërben si gjuhë hyrëse të cilën e njeh Coco/R. Ajo përbëhet nga bashkësia e rregullave gramatikore që përshkruajnë strukturën leksikore dhe sintaksore të gjuhës si dhe përkthimin e saj në gjuhën objekt.

4.3.2.1 Fjalori

Elementet themelore të Coco/R janë identifikatorët, numrat, stringjet dhe konstantat karakter, që janë të përcaktuara si në listingun 4.1.

```
ident = letter {letter | digit}.
number      = digit {digit}.
string      = '"' {anyButQuote} '"'.
char  = '\'' anyButApostrophe '\''.
```

Listingu 4.1 Elementet themelore të Coco/R (sipas [1])

Shkronjat e mëdha dallohen nga të voglat dhe stringjet nuk guxojnë të përshkruhen në më shumë rreshta.

Komentet mbyllen me /* dhe */ dhe mund të jenë të mbivendosur. Ata poashtu mund të fillojnë me // dhe vlejnë deri në fund të rreshtit.

4.3.2.2 Forma e Zgjeruar Backus-Naur (FZBN)

Të gjitha përshkrimet sintaksore në Coco/R janë të shkruar në FZBN – Formën e Zgjeruar Backus-Naur (ang. EBNF – Extended Backus-Naur Form) [2]. Përdoren meta karakterët si në tabelën 4.1.

Tabela 4.1 Meta karakterët për përshkrime sintaksore në Coco/R (sipas [1])

Simboli	Kuptimi	Shembulli	
=	Ndanë anët e produktit	A = a b c .	
.	Përfundon produktin	A = a b c .	
|	Ndan alternativat	a b | c | de	Do të thotë **a b** ose **c** ose **d e**
()	Grupon alternativat	(a | b) c	Do të thotë **a c** ose **b c**
[]	Opsion	[a] b	Do të thotë **a b** ose **b**
{}	Iteracion	{a} b	Do të thotë **b** ose **a b** ose **a a b** ose ...

Atributet shënohen brenda < dhe >. Aksionet semantike janë të mbyllur në (. dhe .). Operatorët + dhe – përdoren për të formuar bashkësi karakteresh.

4.3.3 Struktura e përgjithshme

Përshkrimi i kompilatorit Cocol/R ka strukturën si në listingun 4.2.

```
Cocol =
[Imports]
"COMPILER" ident
[GlobalFieldsAndMethods]
ScannerSpecification
ParserSpecification
"END" ident '.'
```

Listingu 4.2 Struktura e përshkrimit të kompilatorit Coco/R (sipas [1])

Fjala `Cocol` është sinonim për gjuhën e Coco/R, pra nga anglishtja Coco *language*. Vlen të theksohet se gramatika jepet në seksionin `ParserSpecification`, për të cilin dhe seksionet tjerë pason zbërthimi në vijim.

4.3.3.1 Seksioni ScannerSpecification

Skaneri është i përgjegjshëm të "lexojë" tekstin burimor, të mos i përfill simbolet pa kuptim, të identifikojë tokenët dhe të ia kalojë ata parserit. Këto përshkruhen në specifikacionin e skanerit, që përbëhet nga pesë pjesë opsionale si listingun 4.3.

```
ScannerSpecification =
["IGNORECASE"]
["CHARACTERS" {SetDecl}]
["TOKENS" {TokenDecl}]
["PRAGMAS" {PragmaDecl}]
{CommentDecl}
{WhiteSpaceDecl}.
```

Listingu 4.3 Specifikacioni i skanerit në Coco/R (sipas [1])

4.3.3.2 Seksioni ParserSpecification

Specifikimi i parserit është pjesa kryesore e përshkrimit të kompilatorit. Ai përmban produktet e gramatrikës së atribuuar, që përcaktojnë sintaksën e gjuhës që duhet parsuar si dhe përkthimin e tij.

```
ParserSpecification = "PRODUCTIONS" {Production}.
Production = ident [FormalAttributes] [LocalDecl] '=' Expression '.'.
Expression = Term {'|' Term}.
Term = [[Resolver] Factor {Factor}].
Factor = ["WEAK"] Symbol [ActualAttributes]
| '(' Expression ')'
| '[' Expression ']'
| '{' Expression '}'
| "ANY"
| "SYNC"
| SemAction.
Symbol = ident | string | char.
SemAction = "(." ArbitraryStatements ".)".
LocalDecl = SemAction.
FormalAttributes = '<' ArbitraryText '>'.
ActualAttributes = '<' ArbitraryText '>'.
Resolver = "IF" '(' {ANY} ')'.
```

Listingu 4.4 Specifikacioni i parserit në Coco/R (sipas [1])

4.3.3.2.1 Specifikacioni {Production}

Në seksionin për produkte jepen strukturat sintaksore të simboleve joterminale. Produkti përbëhet prej pjesës së majtë dhe të djathtë që ndahen me shenjën e barazimit. Ana e majtë përcakton emrin e joterminalit së bashku me atributet e tij formale dhe variablat lokale të prodhimit. Ana e djathtë përbën shprehjet FZBN që përcaktojnë strukturën e joterminalit si dhe përkthimin e tij në formë të atributeve dhe aksioneve semantike.

Produkutet mund të jepen sipas cilësdo rradhe. Lejohen referncat edhe te joterminalet akoma të padeklaruar. Për çdo terminal duhet të ekzistojë saktësisht një produkt, dhe në veçanti, duhet të ekzistojë produkti për emrin e gramatikës, që është simboli startues i gramatikës.

4.3.3.2.2 Aksionet semantike

Aksioni semantik është pjesë kodi i shkruar në gjuhën që është implementuar Coco/R (në rastin tonë në Java). Ai ekzekutohet nga parseri i gjeneruar në pozitën ku është specifikuar në gramatikë. Aksionet semantike thjeshtë kopjohen në parserin e gjeneruar dhe korrektësia e tyre nuk provohet nga Coco/R. Ata duhet të shënohen brenda shenjave (. dhe .).

4.3.3.2.3 Atributet

Produktet konsiderohen dhe në të vërtetë përkthehen në metoda parsimi. Paraqitja e joterminalit në anën e djathtë të produktit mund të vështrohet si thirrje e metodës parsuese të joterminalit.

Joterminalet mund të kenë atribute, që i korrespondojnë parametrave të metodës parsuese të joterminalit. Atributet mund të jenë hyrëse dhe dalëse. Ato hyrëse i pasohen metodës parsuese kurse ato dalëse kthehen nga metoda parsuese. Atributet shënohen brenda shenjave < dhe > dhe nëse si parametra ka edhe tipa gjenerik si List<T> atëherë përdoren simbolet <. dhe .>

Pasi në Java nuk ka parametra dalës atëherë mund të deklarohet vetëm një atribut dalës, edhe atë ai që kthehet si vlerë nga metoda parsuese. Nëse duhet të kthehen më shumë se një vlerë, si atribut dalës mund të përdoret klasa me atributet përkatëse.

4.3.3.2.4 Konfliktet LL(1)

Parsimi me origjinë rekursive kërkon që gramatika e gjuhës së parsuar të jetë LL(1) që do të thotë e parsueshme nga e majta (ang. Left) në të djathtë me derivime të majta (L) kanonike dhe 1 simbol të vëzhgueshëm përpara. Kjo do të thotë që në çdo

pikë të gramatikës parseri duhet të jetë i aftë që me vëzhgim përpara të vetëm një simboli të vendos të zgjedhë cilën alternativë nga disa që janë në dispozicion.

4.4 CORBA dhe JacORB

CORBA [23][33][34] është teknologji e zhvilluar sipas specifikacionit të OMG (ang. Object Management Group) [35] të cilën e përbëjnë një numër i konsiderueshëm i kompanive dhe institucioneve ndër to të njohura si Microsoft, Hewlett Packard Enterprise, etj. Qëllimi është zhvillimi i specifikimeve dhe produktit që do të mundësonte komunikimin dhe shkëmbimin e informatave ndërmjet objekteve që gjenden në mjedise të ndryshme heterogjene. Heterogjeniteti i mjediseve është karakteristikë e theksuar, pasi që në jetën e përditshme mund të ketë sisteme që punojnë në harduer, sisteme operative dhe të zhvilluara në gjuhë programuese të ndryshme.

CORBA ofron ndërfaqe të pavarura nga platforma dhe modele për aplikacione të distribuuara të orientuara në objekte [33]. Ajo është e pavarur nga gjuha programuese dhe protokollet, që u mundëson sistemeve të zhvillohen në platforma të ndryshme programuese dhe të integrohen.

4.4.1 Karakteristikat e CORBA-s

CORBA është teknologji e distribuuar e orientuar në shërbime. Objektet rezultuese të saj janë të organizuar sipas modelit klient/server. Rëndom, klienti mund të inicojë kërkesë e serveri të përgjigjet. Por, ka raste edhe kur serveri inicon ngjarje, mundësi kjo që është bërë e mundur që nga specifikimet e mëvonshme të CORBA-s. Klienti mund të inicojë kërkesë përmes shtyllave statike ose ndërfaqes dinamike (ang. DII – Dynamic Invocation Interface). Organizmi klient/server në CORBA është paraqitur në figurën 4.5 në vijim.

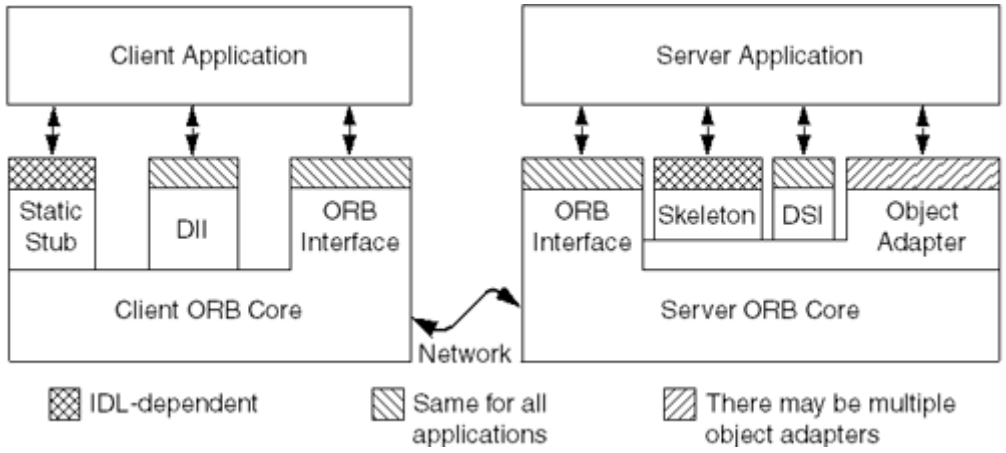


Figura 4.5 Modeli i CORBA-s (sipas [33])

Siç mund të shihet nga figura 4.5 mesazhet e shkëmbyera ndërmjet klientit dhe serverit shkojnë përmes ORB (ang. Object Request Broker). Edhe serveri sikurse edhe klienti mund të zgjedhë nëse kerkesa të i përcillet dhe të përgjigjet përmes skeletit statik ose atij dinamik (ang. DSI – Dynamic Skeleton Interface). Në të dyja rastet si ndërmjetësues ORB – skelet shërben Adapteri i Objektit (ang. Object Adapter).

Duhet përmendur se aktualisht janë të mundura kërkesat sinkrone, asinkrone dhe njëkahëshe (ang. oneway). Në rastin e kërkesës sinkrone, dërgohet kërkesa nga klienti dhe ai bllokohet derisa të i kthehet përgjigjja nga serveri. Në rastin e këkresës asinkrone, klienti e dërgon kërkesën dhe mund të vazhdojë me punë tjera dhe ndërkohë, përmes mekanizmave përkatës, të shikojë nëse ka arritur përgjigjja nga serveri. Në rastin e kërkesës njëkahëshe, klienti nuk pret përgjigje nga serveri për ekzekutim të saj. Në të vërtetë, serveri nuk kthen përgjigje fare për ekzekutim të saj . Për më tepër, ORB-të janë të autorizuar që, nëse kërkesa e tillë do të shkaktonte ngarkesa në rrjet ose performansa tjera negative, të mos ia përcjellë fare serverit përkatës.

4.4.2 Koncepte themelore dhe terminologji në CORBA

Përveç konceptit të klientit dhe serverit të përmendur në paragrafin paraprak përmes skemës, në vijim po i japin edhe disa të tjera shkurtimisht. Po i paraqesim konceptet në vend të shpjegimit të tërësishëm hap pas hapi të vetë CORBA-s pasi që do ta kalonte kornizën e dedikimit të këtij punimi.

4.4.2.1 Gjuha e përcaktimit të ndërfaqes (IDL)

Gjuha e përcaktimit të ndërfaqes (ang. Interface Definition Language) është një nga karakteristikat themelore të CORBA-s. Përdoret për përshkrim të ndërfaqes në bazë të së cilës do të komunikonin klienti dhe serveri. Përdoret nga kompilatori për të gjeneruar ndërfaqet dhe kodin tjetër (shtyllat, skeletët, etj.) të nevojshëm për aplikacionin. Është a pavarur nga gjuha programuese, kështu që mundëson komunikimin e klientit të shkruar në një gjuhë programuese me serverin të shkruar në gjuhë tjetër programuese. Është gjuhë e pastër deklarative, kështu me të nuk mund të implementohet asgjë sa i përket ekzekutimit të instruksioneve ose gjendjeve të objektit.

Siç e kemi përmendur në paragrafin 3.5 sintaksa e saj përshkruhet me rregulla gramatikore të FZBN dhe për detaje të plota mund të i referoheni literaturës [31]. Gramatika e reduktuar e kompilatorit (shih listingun 3.1 në paragrafin 3.5) tonë përcaktohet me vetëm 16 rregulla gramatikore. Këto mjaftojnë për të përshkruar funksionalitete të sensorëve dhe aktuatorëve përmes serverit të implementuar në mikrokontrollor. Për shembull, me rregullat e përcaktuara mund të përshkruhet funksionaliteti i paraqitur në listingun 4.5 në vijim.

```
exception DeviceNotFound {short deviceID;};

interface TempSensor {
      readonly attribute long minTempRead;
      readonly attribute long maxTempRead;
      float readTemp(in short devID) raises(DeviceNotFound);
};

interface Termostat {
      readonly attribute long minTempAllowed;
      readonly attribute long maxTempAllowed;
      exception TempNotAllowed {short deviceID; float temperature;};
      void setTemperature(in short devID, in float temp) raises
(DeviceNotFound, TempNotAllowed);
};
```

Listingu 4.5 Përshkrimi i sensorit të temperaturës dhe termostatit në gjuhën IDL përmes ndërfaqeve përkatëse

Në listingun 4.5 është paraqitur përshkrimi i ndërfaqeve të sensorit të temperaturës dhe termostatit. Në fillim është deklaruar gabimi global `DeviceNotFound` që mund të paraqitet në të dyja ndërfaqet. Poashtu në ndërfaqen

`Termostat` është deklaruar gabimi `TempNotAllowed` që është specifik për atë ndërfaqe. Në listingun 4.5, ndërfaqet kanë atribute që mund vetëm të lexohen, edhe pse nën-gramatika jonë lejon të deklarohen edhe ata që mund të u vendoset vlera. Kuptimi i atributeve është evident nga emërtimi i tyre. Gabimet kthejnë edhe vlera të tipit themelor `short`. Operacionet poashtu pranojnë dhe kthejnë vlera të tipave themelor, `short` dhe `long` respektivisht. Operacioni i dytë kthen `void`, që do të thotë nuk kthen asgjë.

4.4.2.2 ORB

ORB (ang. Object Request Broker) siç tregon edhe emri merret me transport dhe dorëzim të mesazheve ndërmjet objekteve në CORBA (shih figurën 4.5). Përmes ORB objekti që inicon kërkesë (klienti) nuk duhet të brengoset për vendin e objektit tjetër që pranon kërkesën (serverit), qoftë ai në distancë në rrjet ose lokal. Inicuesi i kërkesës duhet vetëm ta dijë Referencën e Objektit (shih paragrafin 4.4.2.5) ku është i hostuar objekti që pranon kërkesën, dhe të krijojë mesazhin konform IDL që bëhet përmes shtyllave që gjenerohen nga IDL kompilatori për gjuhën e dhënë programuese. Me këtë rast objekti që pranon kërkesën i duket si lokal inicuesit të kërkesës. Kjo është e njohur si transparencë e vendit dhe qasjes [36].

4.4.2.3 Objekt Adapteri

Objekt adapteri që përdoret tani është Objekt Adapteri i Bartshëm (ang. POA – Portable Object Adapter) që është pasues i atij themelor (ang. BOA – Basic Object Adapter). Specifikacioni i objekt adapterit themelor ishte lënë i mangët, me qëllim që të i shtohen funksionalitete në varësi nga platforma e implementimit. Si rezultat i kësaj, shumë prodhues implementonin pjesë të ndryshme të BOA-s me ndryshime në semantikë të saj. Kjo përvojë shërbeu si bazë për standardizimin e POA-s. Karakteristikat e POA-s përshkruhen përmes IDL. Roli i POA-s është krijimi i objektit, regjistrimi i servantit dhe dërgimi i kërkesës. Kështu POA ofron një bashkësi ndërfaqesh standarde për menaxhim të referencave të objekteve dhe servantëve.

4.4.2.4 GIOP (IIOP)

GIOP (ang. General Inter Orb Protocol) paraqet platformë, mbi të cilin do të duhej të ndërtoheshin protokollet që do të mundësonin komunikim ndërmjet objekteve CORBA, respektivisht ORB-ve siç tregon edhe vetë shkurtesa. Konkretisht, në bazë të tij është ndërtuar protokolli IIOP (ang. Internet Inter Orb Protocol) me të cilin komunikojnë ORB-të në Internet dhe rrjeta lokale. GIOP përcakton mekanizmat e transportit, paraqitjen e të dhënave (ang. CDR – Common Data Representation) dhe formatet e mesazheve. Transporti merret me lidhjet si dhe shkëmbimin e bajtëve ndërmjet ORB-ve. Paraqitja e të dhënave ka të bëjë me rradhitjen e të dhënave, konkretisht të dhënat me tipat përkatës në serinë e bajtëve që shkëmbehen, duke përfshirë këtu nëse biti (bajti) me peshë më të madhe të vendoset në adresë më të lartë ose më të ulët (konceptet big-endian dhe little-endian). Formatet e mesazheve përcaktojnë tipat e mesazheve dhe ndërtimin konkret të tyre. Janë të përcaktuar disa lloje të mesazheve që mund të shkëmbejnë klienti dhe serveri, por për funksionim minimal janë të nevojshme vetëm dy lloje të tyre: kërkesa (ang. request) dhe përgjigjja (ang. response). Kërkesa inicohet nga klienti kurse përgjigjja nga serveri. IIOP përmban të dhëna që janë specifike për rrjetat, si: hosti, porta, çelësi i objektit etj.

4.4.2.5 IOR

IOR (ang. Interoperable Object Reference) në CORBA shërben për ta identifikuar objektin e caktuar në mënyrë unike. Kjo i shërben ORB-ve për ta lokalizuar objektin e caktuar dhe dërguar mesazhe. Çdo implementim CORBA ka mekanizmat përkatës për manipulim me IOR: krijim të tyre në bazë të objektit të dhënë, paraqitje/ruajtje në formë të përshtatshme dhe përdorim respektivisht thirrje të tyre. Duhet të theksohet se IOR është i "padukshëm" për klientin dhe në të ruhen të gjitha informatat për lokalizim (transporti, protokolli dhe çelësi) të objektit dhe dërgim të mesazheve. Me të "padukshëm" nënkuptojmë se ai nuk mund të lexohet me sy siç p.sh. është rasti i adresave të http objekteve. IOR ruhet në skedar tekstual i koduar si seri e bajtevë heksadecimalë (shih listing 4.6). Në një IOR të vetëm mund të ruhen të dhëna për të kapur objektin nga protokolle te ndryshëm.

```
IOR:000000000000001549444C3A546573744D495350436F6E6E3A312E30000000000
000000100000000000000050000102000000000E3139322E3136382E302E3130310005
340000001E333431353531383937382F000B303517300628100630463814141B484C1
B000000000001000000000000000800000000004A414300
```

Listingu 4.6 Shembull ilustrimi i IOR i rujatur në skedar tekstual

4.4.3 JacORB

JacORB [30][38] është implementim i specifikacionit CORBA në gjuhën programuese Java. Pjesën eksperimentale ne e kemi realizuar me versionin 3.4 të JacORB, edhe pse kur është shkruar ky disertacion ka qenë në dispozicion versioni 3.7. Sa i përket realizimeve tona nuk ka dallim versioni 3.7 në krahasim me atë 3.4.

4.4.3.1 Instalimi

Për ta përdorur JacORB 3.4 duhet që në kompjuter të jetë e instaluar JDK 1.6 ose më e re. Për ta kompiluar vetë JacORB-in, shembujt përcjellës dhe projektet që do ti krijojmë duhet të jetë e instaluar vegla për kompilim/ekzekutim e bazuar në XML e quajtur Ant. Versionet më të reja të JacORB në vend të Ant e kërkojnë veglën Maven.

Instalimi është i thjeshtë. Shkarkohet skedari ZIP nga sajti zyrtar i JacORB [30] dhe dekomprimohet në JACORB_HOME (te ne është JACORB_HOME=C:\jacorb-3.4). Nëse duam ta ri-kompilojmë JacORB-in, shkarkojmë versionin ZIP të kodeve burimore (jacorb-3.4-source.zip) dhe e dekomprimojmë në JACORB_HOME. Pastaj nga rreshti komandues (ang. command prompt) shkojmë ne direktorinë JACORB_HOME dhe ekzekutojmë komandën ‘ant’. Nëse duam ta përdorim JacORB-in tanimë të kompiluar, shkarkojmë versionin ZIP binar (jacorb-3.4-binary.zip) dhe e dekomprimojmë në JACORB_HOME. Pas të dy këtyre veprimeve vendosen si variabla sistemore (ang. system variables) variablat JACORB_HOME dhe JACORB_HOME\bin.

4.4.3.2 Krijimi i projektit të ri

Pas instalimit të rrethinës JacORB hapi vijues është krijimi i projekteve respektivisht aplikacioneve. Për këtë duhet ndjekur rrugën e projekteve demonstrative

që vijnë në pakon e softuerit JacORB. Projektet demonstrative gjenden në direktorinë JACORB_HOME\demo.

Në direktorinë JACORB_HOME e krijojmë atë myprojects, pra krijojmë direktorinë JACORB_HOME\myprojects. Në JACORB_HOME\myprojects krijojmë direktorinë e projektit sipas dëshirës p.sh. nëse i referohemi përkufizimit IDL në listingun 4.5 krijojmë direktorinë TermoProject pra JACORB_HOME\myprojects\TermoProject. Nga ndonjëri projekt demonstrativ kopjojmë skedarin build.xml p.sh. nga direktoria JACORB_HOME\demo\hello dhe e vendosim në direkotoriumin e projektit tonë (JACORB_HOME\myprojects\TermoProject). Më pastaj skedarin e kopjuar e modifikojmë, ashtu që të duket si vijon:

```
<?xml version="1.0"?>

<project name="JacORB TermoProject myprojects" default="myprojects"
basedir=".">

    <import file="../common/common-demo.xml" />

    <target name="myprojects" depends="compile">
        <run-demo>
            <run-server>
                <run-default-server classname="TermoProjectServer" />
            </run-server>

            <run-client>
                <run-default-client classname="TermoProjectClient" />
            </run-client>
        </run-demo>
    </target>

</project>
```

Listingu 4.7 Skedari build.xml specifik për projekt

Pjesët që duhet modifikuar në skedarin e kopjuar build.xml për të ia përshtatur projektit tonë janë paraqitur me shkronja të **`trasha`**.

Më tej, krijojmë direktoritë idl dhe src në direktorinë e projektit tonë, pra krijojmë JACORB_HOME\myprojects\TermoProject\idl dhe JACORB_HOME\myprojects\TermoProject\src. Në direktorinë idl e vendosim skedarin IDL nga listingu 4.5 dhe e emërtojmë server.idl. Në direktorinë src i vendosim skedarët e implementimit të ndërfaqeve TempSensorImpl.java dhe

TermostatImpl.java si dhe skedarët e klientit respektivisht serverëve (instancuesve të TempSensorImpl.java dhe TermostatImpl.java). Skedarët e klientit dhe serverit janë specifik për aplikacion. Klienti mund të thërrasë operacione të një apo më shumë instancave të implementimit të ndërfaqeve.

Hapi vijues është gjenerimi respektivisht kompilimi që është i automatizuar dhe bëhet duke e ekzekutuar komandën 'ant compile' nga rrethina komanduese (ang. command prompt) në direktorinë JACORB_HOME\myprojects\TermoProject. Skedarët që gjenerohen dhe kompilohen me këtë rast, si dhe roli i tyre janë paraqitur në paragrafin vijues (4.4.3.3).

Pas kompilimit fillimisht duhet të startohet serveri. Menyra më e rëndomtë është nga rreshti komandues, duke shkuar në direktorinë JACORB_HOME\myprojects\TermoProject\build\classes dhe pastaj ekzekutuar komandën 'jaco TermoProjectServer emer.ior'. Komanda 'jaco' gjendet në JACORB_HOME\bin dhe paraqet beç (ang. batch) skedarin (skriptën) për thirrje të makinës virtuale 'java' me parametra shtesë karakteristik për JacORB. Parametri 'TermoProjectServer' paraqet klasën e kompiluar të serverit i cili në brendi instancon klasën implementuese të ndërfaqes (krijon servisin), që si parameter hyrës përmban 'emer.ior' ('emer.ior' mund të jetë emërtim çfarëdo legal për skedar) – vendin për IOR të klasës implentuese të ndërfaqes në fjalë. Nëse ka më shumë se një ndërfaqe të implementuar (si në rastin e skedarit IDL të përcaktuar në listingun 4.5), atëherë është e përshtatshme të krijohet nga një server klasë që instancon klasën implementuese të secilës ndërfaqe, dhe të startohen veçmas duke dhënë në dalje skedarë IOR të veçantë për secilën ndërfaqe të implementuar. Komanda përkatëse p.sh. për ndërfaqen TempSensor do të mund të ishte 'jaco TermoProjectTempSensorServer tpsTempSensor.ior'. Nëse ka më shumë se një server atëherë ekzekutimi i tyre të të ishte 'jaco TermoProjectTempSensorServer tpsTempSensor1.ior' ose 'jaco TermoProjectTempSensorServer tpsTempSensor1.ior prm2 prm3 ...', ku prm2, prm3 janë parametrat shtesë që duhet të i përcillen serverit për ta dalluar atë, respektivisht për ta adresuar sensorin tjetër.

Edhe klienti do të startohej nga direktoria JACORB_HOME\myprojects\TermoProject\build\classes duke ekzekutuar komandën

‘jaco TermoProjectClient emer.ior’, ku siç mund të shihet ai e përdor si hyrje skedarin IOR ‘emer.ior’ për ta lokalizuar serverin përkatës. Nëse klienti dhe serveri gjenden në vende (kompjuterë) të ndryshëm, atëherë, skedari IOR pasi të jetë gjeneruar duke startuar serverin përkatës, do të mund të i dërgohej klientit (p.sh. përmes emailit) dhe përdorej nga ai siç është dhënë më parë.

4.4.3.3 Skedarët e gjeneruar nga kompilatori JacORB IDL

Pas ekzekutimit të komandës ‘ant compile’ (shih paragrafin paraprak 4.4.3.2) në direktorinë JACORB_HOME\myprojects\TermoProject\build\generated gjenerohen disa skedarë të nevojshëm për punë normale të serverit dhe klientit. Skedarët gjenerohen në bazë të përmbajtjes së skedarit (skedarëve) IDL të përcaktuar. Në rastin tonë, gjenerohen nga disa skedarë për ndërfaqet TempSensor dhe Termostat, gabimin global DeviceNotFound si dhe gabimin lokal TempNotAllowed të përcaktuar brenda ndërfaqes Termostat. Roli i skedarëve të gjeneruar është si vijon.

Për ndërfaqen Termostat. Skedarët Termostat.java, TermostatHelper.java dhe TermostatHolder.java përdoren direkt nga zhvilluesi i aplikacionit. Skedari Termostat.java përmban ndërfaqe në Java që kombinon operacionet e pasqyruara së bashku me specifikacionin CORBA të funksionalitetit. Klasa TermostatHelper (e përcaktuar në TermostatHelper.java) ofron funksionet (operacionet) e shërbimeve, më të rëndomtin narrow() të përdorur për ta kthyer referencën si tip të objektit. Klasa TermostatHolder (TermostatHolder.java) përdoret si kontejner për ti marrë argumentët "out" nga operacioni. Skedarët TermostatPOA.java dhe TermostatPOATie.java përdoren në implementim të objektit CORBA. Klasa TermostatPOA (TermostatPOA.java) është klasë abstrakte bazë që ofron lidhje ndërmjet ORB infrastrukturës në anën e serverit dhe kodit servant të aplikacionit. Në këtë rast kodi servant zgjeron klasën TermostatPOA, duke ofruar implementacionin të përcaktuar me operacionet e Objektit. TermostatPOATie (TermostatPOATie.java) e kryen funksionin e njëjtë si TermostatPOA, por pranon çfarëdo klasë aplikacioni që ofron operacionet e Objektit si delegat për servantin. Në këtë rast mund të kemi një objekt servant në Java të vetëm që implementon më shumë ndërfaqe, të përdorur si delegat të ndarë. Skedari TermostatOperations.java përmban ndërfaqe në Java që i përfshin të gjithë operacionet e pasqyruar të përcaktuar në skedarin IDL. Ajo nuk

përdoret drejtpërdrejt nga kodi i aplikacionit, përveç kur implementohet nga servanti. Skedari _TermostatStub.java përmban elemente të ORB infrastrukturës të domosdoshëm nga ana e klientit për të i dërguar kërkesa objekteve në distancë. Objekti i ngushtuar CORBA implementohet nga kjo shtyllë. Derisa shtylla ekziston në hapsirën adresuese të klient apliakcionit, aplikacioni do të duhej të i përdorte vetëm operacionet e përcaktuara në IDL.

Për gabimin global DeviceNotFound, roli i skedarëve DeviceNotFound.java, DeviceNotFoundHelper.java dhe DeviceNotFoundHolder.java është i ngjashëm sikur te ndërfaqja e përshkruar më parë. Për dallim nga ndërfaqja, për gabimin nuk ka POA dhe _stub klasë pasi që nuk ka operacione të shoqëruar me gabime.

Për ndërfaqen TempSensor, shih përshkrimin për ndërfaqen Termostat pasi që vlen e njëjta.

Për gabimin lokal TempNotAllowed të përcaktuar brenda ndërfaqes Termostat, gjenerohen skedarët e njëjtë sikur për gabimin DeviceNotFound. Vetëm se në këtë rast, ato gjenerohen në direktorinë JACORB_HOME\myprojects\TermoProject\build\generated\TermostatPackage, pra në direktori të veçantë TermostatPackage të përcaktuara brenda pakos TermostatPackage, që është specifike për ndërfaqen Termostat.

4.4.3.4 Mapimi i tipave IDL – Java

Në tabelën 4.2 në vijim janë paraqitur pasqyrimet e tipave bazikë të parametrave IDL në Java.

Tabela 4.2 Mapimi i tipave bazikë IDL në Java

Tipi në IDL	Tipi në Java	Klasa Holder
octet	byte	org.omg.CORBA.ByteHolder
char	char	org.omg.CORBA.CharHolder
boolean	boolean	org.omg.CORBA.BooleanHolder
short	short	org.omg.CORBA.ShortHolder
unsigned short	short	org.omg.CORBA.ShortHolder
long	int	org.omg.CORBA.IntHolder
unsigned long	int	org.omg.CORBA.IntHolder
float	float	org.omg.CORBA.FloatHolder

long long	long	org.omg.CORBA.LongHolder
unsigned long long	long	org.omg.CORBA.LongHolder
double	double	org.omg.CORBA.DoubleHolder
long double	double	org.omg.CORBA.DoubleHolder

Ndërsa në tabelën 4.3 janë dhënë pasqyrimet e tipave të tjerë të dhënave IDL në Java [39], por vetëm ata që përkrahen nga sintaksa jonë e modifikuar e paraqitur në listingun 3.1.

Tabela 4.3 Mapimi i tipave të tjerë IDL në Java

Tipi në IDL	**Tipi në Java**
Ndërfaqe	Ndërfaqe nënshkrimi dhe të operacioneve, klasë ndihmëse dhe mbajtëse
Gabim	Klasë
Atribut vetëm i lexueshëm	Metodë e qasjes
Atribut i lexuëshëm/modifikueshëm	Metoda të qasjes dhe modifikimit
Operacion	Metodë e qasjes

Është e qartë p.sh. për ndërfaqe të cilat janë ndërfaqet e nënshkrimit dhe operacioneve nga paragrafi 4.4.3.3 për skedarët e gjeneruar nga kompilatori IDL.

KAPITULLI 5

Realizimet – strukturat e kodit dhe gjeneratorët

5.1 Hyrje

Në këtë kapitull janë paraqitur realizimet konkrete të propozimeve për kontribut të paraqitur në kapitullin 3. Kontributi rezulton në ndërtimin e një gjeneratori të kodit burim të nevojshëm për integrimin e pajisjeve të mbyllura të kufizuara, konkretisht të mikrokontrollorëve 8051, në Arkitektura të Orientuara në Shërbime (AOSh). Si përfaqësuese konkrete të AOSh janë marrë teknologjitë CORBA dhe Shërbimet Ueb. Kështu, gjenerohet kodi për integrim të mikrokontrollorit 8051 në implementimet në Java të teknologjive përkatëse JacORB (për CORBA) dhe JAX-WS (për Shërbime Ueb). E kemi theksuar në kapitullin 3 ndarjen e përgjegjësive në mes të mikrokontrollorit dhe kompjuterit personal si pajisje të fuqishme. Mikrokontrollori ekzekuton versionin e thjeshtuar të shërbimit ndërsa në kompjuter ekzekutohet serveri lidhës që paraqet ndërmjetësues në mes të klientit dhe serverit në mikrokontrollor. Prandaj, gjenerohen dy lloje kodesh, ai që ka për t'u ekzekutuar në mikrokontrollor dhe ai në server lidhës. Kodi i mikrokontrollorit gjenerohet në asembler (C51ASM [26]), ndërsa kodi i serverit lidhës gjenerohet në gjuhën Java konform modaliteteve të teknologjisë JacORB respektivisht JAX-WS. Fillimisht është paraqitur struktura e kodit ose organizimi i funksionimit brenda skedarëve përkatës për të vazhduar me përshkrimin e gjeneratorëve që e gjenerojnë atë kod.

5.2 Struktura e kodit të gjeneruar

Struktura e kodit nënkupton organizmin e kodit brenda skedarëve duke përshkruar funksionalitetin e pjesëve të caktuara. Gjenerohen dy lloje kodesh. Kodi që ka për t'u ekzekutuar në mikrokontrollor i njohur si kodi i servisit si dhe kodi i serverit lidhës që ka për t'u ekzekutuar në kompjuter. Kodi i servisit gjenerohet në asembler për 8051, ndërsa kodi i serverit lidhës në Java konform logjikës dhe librarive

të kërkuar nga JacORB (për CORBA) dhe JAX-WS (për Shërbime Ueb). Do ta paraqesim strukturën me blloqe dhe pseudokod për ta kuptuar funksionin si modul dhe ekuivalentin përkatës në Asembler dhe Java, respektivisht. Poashtu do të jepen edhe përshkrimet e rolit të dy klasëve ndihmëse të serverit lidhës, `MISPSerialManaged` dhe `StreamOperations2`. `MISPSerialManaged` përdoret për realizim të komunikimit serial me mikrokontrollorin (shkëmbim të mesazheve), ndërsa `StreamOperations2` për ndërtim dhe manipulim me mesazhe.

Analiza e strukturës së kodit të gjeneruar bëhet për IDL të përcaktuar në listingun 4.5 dhe skedar të implementimit të dhënë në listingun 5.1 në vijim.

```
implement TempSensor;
implement Termostat;
```

Listingu 5.1 Skedari i implementimit të ndërfaqeve

5.2.1 Struktura e serverit 8051

Kodi i serverit 8051 gjenerohet në një skedar të vetëm. Në përgjithësi struktura e kodit të gjeneruar të mikrokontrollorit 8051 mund të paraqitet përmes blloqeve në figurën 5.1 në vijim.

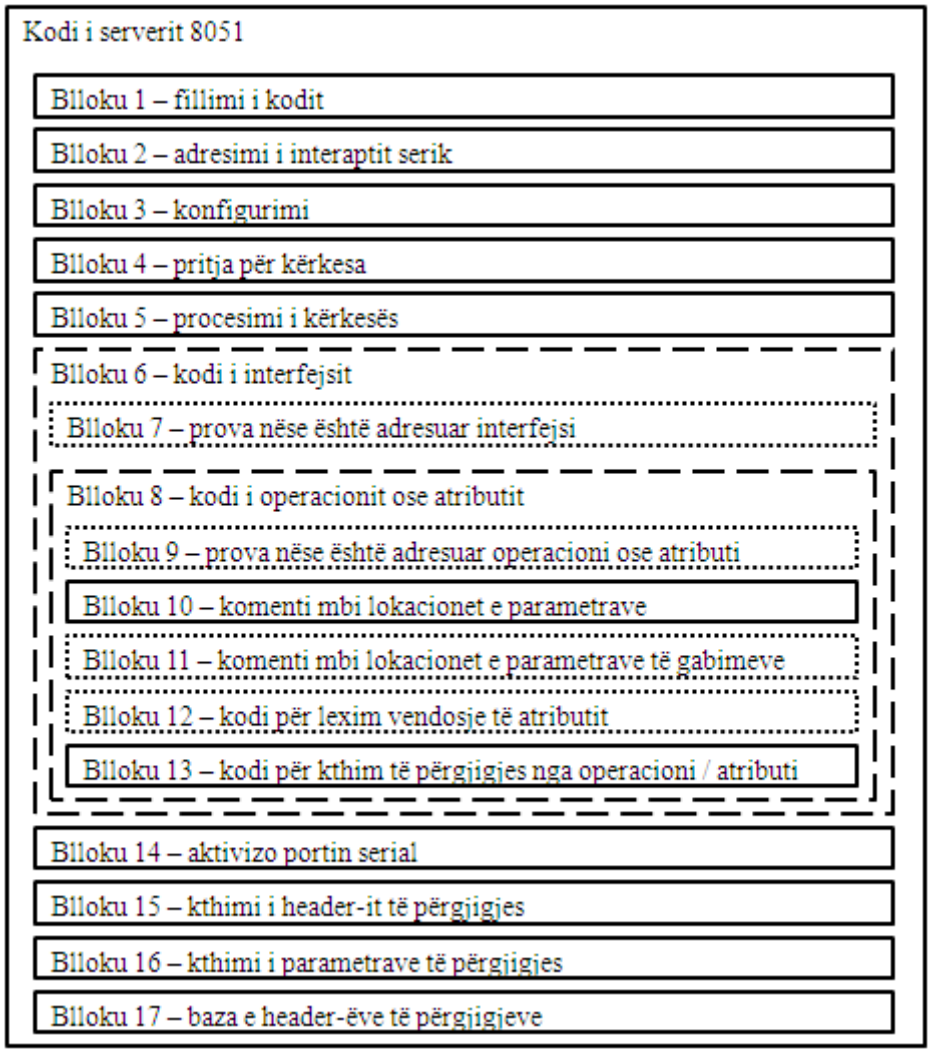


Figura 5.1 Kodi i serverit të realizuar në mikrokontrollorin 8051 përmes blloqeve

Në figurën 5.1 mund të shihen blloqe me vijë të plotë të pandërprerë (si p.sh. blloku 1) që do të thotë se ato blloqe kodi paraqiten gjithmonë. Blloqet me vijë të ndërprerë (si p.sh. blloku 6) mund të mos paraqiten ose të paraqiten një e më shumë herë. Ndërsa blloqet me pika (si p.sh. blloku 7) mund të mos paraqiten ose të paraqiten vetëm një herë.

Kodi i bllokut 1 është i thjeshtë dhe përbëhet prej dy instruksioneve (shih shtojcën A.1). Instruksioni i parë ORG paraqet direktivë që i tregon asemblerit ku të fillohet me shkrimin e kodit në vijim. Kështu instruksioni vijues `JMP InitSerComAndEnblSerInt` shkruhet në adresën 0000H. Kjo do të thotë që në

fillim të kodit të vazhdohet me ekzekutim në adresën e përcaktuar me labelën `InitSerComAndEnblSerInt`.

I ngjashëm është edhe kodi i bllokut 2 (shih shtojcën A.2). Në këtë rast në adresën 0023H shkruhet instruksioni për kërcim në adresën e përcaktuar me labelën SPISR. Adresa 0023H është adresa në të cilën vazhdohet me ekzekutim nëse ndodh interapt i portës seriale (në këtë rast kur arrin bajti i parë në portën seriale).

Blloku 3 është paksa më kompleks dhe kodi i tij është si në shtojcën A.3. Kuptimi i dy rreshtave të parë, `ORG 0030H` dhe labelës `InitSerComAndEnblSerInt` është i qartë. Vlera 0030H i korrespondon vlerës prej ku në mikrokontrollorin 8051 mund të shkruhet kodi i përdoruesit. Adresat më të vogla se 0030H deri në 0000H janë të rezervuara për përpunim të interapteve.

Instruksioni `MOV SCON, #50H` shërben për konfigurim të portës seriale. Secili nga bitët e vlerës heksadecimale të bajtit 50H të shprehur në formë binare (01010000) ka kuptim të caktuar. Tre bitët e parë përcaktojnë modën e punës së portës seriale. Dy bitët e parë përcaktojnë modën gjatë punës një-mikroporcesorike kurse biti i tretë modën e punës shumë-procesorike në mikrokontrollor. Pasi që biti i tretë është 0 (moda njëprocesorike) kuptimi i dy bitëve të parë është se porta seriale punon në modën 1, që tregon se frekuenca e punës së portës seriale është variabile e përcaktuar nga tajmeri 1, respektivisht instruksioni në vijim `MOV TH1, #239`. Biti i katërt shërben për aktivizim të portës seriale, që është i qasshëm edhe si i veçantë përmes regjistrit të adresueshëm si bit REN. Të rëndësisë janë edhe dy bitët e fundit, të adresueshëm si bit TI dhe RI. Kuptimi i tyre do të jepet në vijim kur përdoren në kod.

Instruksioni `MOV TMOD, #20H` përdoret për caktim të modit të tajmerëve. Katër bitët e parë (0010) shërbejnë për tajmerin 1 ndërsa katër të tjerët (0000) për tajmerin 0. I rëndësisë për ne është tajmeri 1. Biti i parë (0) tregon se tajmeri punon pavarësisht vlerës në pinin $\overline{INT1}$. Biti i dytë (0) tregon se përdoret për matje intervali (e jo si numrues nëse është 1). Dy bitët e fundit (10) përcaktojnë modën e punës e që në këtë rast e ka kuptimin se tajmeri punon në modën 2 me vet-ngarkim (ang. auto-reload). Ky mod shërben për gjenerim të frekuencës së ndryshueshme, të caktuar me

vlerën e bajtit me peshë më të madhe të numruesit të tajmerit, të përcaktuar me instruksionin vijues `MOV TH1, #239`. Quhet me vet-ngarkim, pasi kjo vlerë decimale (#239) numron deri në kalimin 255 në 0, me ç'rast gjenerohet një impuls frekuenca e të cilit shërben si clock për ngasje të portës seriale. Lidhja në mes frekuencës së portës seriale (f_s) dhe vlerës së bajtit TH1 varet edhe nga frekuenca e punës së clock-ut (f_c) të mikrokontrollorit dhe jepet përmes formulës matematikore:

$$TH1 = 256 - \left[\frac{f_c}{f_s \cdot 384}\right]$$

Shenjat [ ] paraqesin vlerën e rrumbullakësuar të shprehjes brenda tyre.

Instruksioni `SETB TR1` starton tajmerin 1, me ç'rast ai bëhet i gatshëm për punë. Instruksioni `SETB PS` ia cakton interaptit të portës seriale prioritetin më të lartë. Nëse ndodhë edhe ndonjë interapt tjetër përveç atij të portës seriale ky i fundit përpunohet i pari. Instruksioni `SETB ES` aktivizon përpunimin e interaptit të portës seriale. Instruksioni `SETB EA` aktivizon bitin global të përpunimit të interapteve. Pa vlerën 1 të këtij biti asnjë interapt nuk përpunohet.

Kodi i bllokut 4 është dhënë në shtojcën A.4. Labela MAIN tregon në adresën e instruksionit vijues `JMP $`, i cili përmes shenjës $ tregon se kërcimi duhet të bëhet në adresën e po atij instruksioni. Pra, programi kryesor sillet në të njëjtin vend dhe pret për kërkesë nga porta seriale, përpunimi i së cilës inicohet përmes interaptit të portës seriale. Këtu, në vend të instruksionit `JMP $`, mund të shkruhet ndonjë program të cilin mund ta ekzekutojë mikrokontrollori derisa nuk ka kërkesa nga jashtë. Skenar tipik mund të jetë rregullimi (mbajtja) e një vlere të caktuar në dalje e cila caktohet përmes atributit të përcaktuar në ndërfaqe.

Kodi i bllokut 5 është dhënë në shtojcën A.5. Për të mënjanuar konfuzionin rreshtat e kodit janë shënuar me numra. Rreshti 1 paraqet labelën (treguesin) ku kërcëhet nga blloku 2. Blloku 2 paraqet adresën ku përpunohet kodi i interaptit serial dhe prej aty vazhdohet (kërcehet) në bllokun aktual në fjalë 5, për arsye se kodi i përpunimit është i madh dhe tejkalon adresat origjinale të rezervuara për përpunim të interaptit serial. Në rreshtin 2, adresën e të cilit e tregon labela `SPISR` bëhet deaktivizmi i interapteve. Kjo për arsye se për çdo bajt që lexohet/dërgohet gjenerohet

interapt serial dhe kështu do të na kthente në fillim të përpunimit të interapteve, kurse neve na duhet leximi i bajtëve të tjerë, gjë që bëhet me instruksionet në vijim.

Në rreshtin 4 pritet derisa të bëhet vlera e regjistrit RI e barabartë me 1, e cila ishte inicializuar ne bllokun 3 me vlerën 0. Kjo do të thotë se mikrokontrollori duhet të presë derisa të lexohet i tërë bajti nga porta dhe të vendoset në regjistrin SBUF. Në rreshtin 5 vlera e regjistrit RI bëhet zero, që do të thotë se porta është serish i gatshme për lexim të bajtit të ri. Në rreshtin 6, vlera e bajtit e lexuar nga porta seriale (SBUF) ruhet në akumulatorin A për manipulim të mëtejshëm. Kjo vlerë paraqet gjatësinë e mesazhit, duke e përfshirë edhe "vetëveten", të përcaktuar në përputhje me protokollin AMISP. Në rreshtin 8 regjistrit R0 i ndahet vlera 144 (e llogaritur nga gjeneratori i kodit) e cila përdoret nga instruksioni në rreshtin 9, për ta ruajtur vlerën e akumulatorit A në vendin (144) në kujtesën RAM të treguar nga regjistri R0 përmes adresimit indirekt. Në rreshtin 11 vlera e akumulatorit A zvogëlohet për 1. Kjo vlerë tregon numrin e bajtëve që kanë mbetur për t'u lexuar. Në rreshtin 13 provohet nëse vlera e akumulatorit A është e ndryshme prej 0, ose shprehur me fjalë nëse kanë mbetur ende bajtë për t'u lexuar, që në të shumtën e rasteve ndodhë rëndom. Nëse A është e ndryshme prej 0 vazhdohet me ekzekutim/kërcim në labelën `ContReadStreamBytes` ose rreshtin 15. Nëse A është 0 vazhdohet në rreshtin 14 i cili ka të bëjë me kërcim në labelën `ClrRenAndProc` ose rreshtin 25. Rreshti 15 tregon adresën e instruksionit në rreshtin 16. Në rreshtin 16 ruhet vlera e akumulatorit A në regjistrin R1. Ky regjistër (R1) shërben si numrator për bajtët e mbetur për t'u lexuar nga porta seriale. Në rreshtin 18 përmes labelës `RecvIncStreamBytes` formohet cikli i leximit të bajtëve. Kështu, në rreshtin 19 vlera e regjistrit R1 rritet për 1, që tregon vendin e ardhshëm në kujtesën RAM ku ka për t'u ruajtur bajti i lexuar nga porta seriale. Rreshtat 20 dhe 21 kanë rolin e rreshtave 4 dhe 5 respektivisht, të lartpërmendur. Në rreshtin 22 bajti i lexuar nga porta (`SBUF`) ruhet në vendin e treguar nga regjistri R0 përmes adresimit indirekt. Në rreshtin 23 vlera e regjistrit R1 zbritet për një, dhe, nëse kjo vlerë është e ndryshme prej zero, që do të thotë se kanë mbetur ende bajtë pa u lexuar (konform AMISP), atëherë kërcehet në adresën e treguar me labelën `RecvIncStreamBytes` (rreshti 18). Në rreshtin 25 është labela ClrRenAndProc që tregon në instruksionin në rreshtin 26, e që poashtu arrihet si

kërcim nga rreshti 14 nëse nuk ka më bajtë për t'u lexuar. Në rreshtin 26 mbyllet ose deaktivizohet leximi nga porta seriale, pasi paraprakisht janë lexuar të gjithë bajtët e kërkesës. Në vijim pason kodi për përpunim (identifikim) të kërkesës, implementim të operacioneve dhe kthim të përgjigjes poashtu përmes portës seriale.

Blloku 6 ose blloku i ndërfaqes, përbëhet nga blloqe të gjithë opsionalë. Ky bllok gjenerohet për secilën nga ndërfaqet e implementuara përmes skedarit të implementimit (shih listingun 5.1).

Blloku 7 është opsional që provon nëse është adresuar ndërfaqja e caktuar. Kodi i tij duket si në shtojcën A.6. Në rreshtin e parë regjistrit R0 i ndahet vlera 145 decimale e gjeneruar nga parallogaritja e vendit të pritur të adresës së vendit të ndërfaqes. Në rreshtin vijues nga ky vend përmes adresimit indirekt lexohet vlera dhe i ndahet akumulatorit A. Në rreshtin vijues është gjeneruar labela nga e cila fillon prova ose vihet si kërcim nga blloku i njejtë, nëse është adresuar ndërfaqja e dhënë me ID 0. Labela fillon me tekstin `ChckIntID` nga anglishtja Check Interface ID. Teksti vijues i ndarë me simbolin _ tregon ndërfaqen për të cilin bëhet prova dhe vlera 0 në fund tregon instancën. Në rreshtin vijues provohet nëse nuk është adresuar ndërfaqja me numër identifikues (ID) 0. Nëse nuk është adresuar ndërfaqja 0 kërcehet në labelën `ChckIntID_TempSensor_0_Failed` (rreshti i dytë vijues), e cila tregon në instruksionin `JMP ChckIntID_Termostat_1` (rreshti i tretë vijues). Pra vazhdohet në bllokun e ngjashëm që provon nëse është adresuar ndërfaqja vijuese, me ID 1. Nëse nuk ka ndërfaqe vijuese, që do të thotë se ky që e kemi provuar ka qenë i fundit, atëherë kërcehet në labelën `EnSerComAndInt`. Nëse është adresuar ndërfaqja në fjalë (me ID 0), atëherë vazhdohet ekzekutimi në rreshtin e parë vijues që ka instruksionin `JMP Chck_TempSensor_0_operationIDs`. Ky instruksion është kërcim në bllokun ku bëhet provimi i adresimit (përpunimi) të operacioneve të caktuar brenda ndërfaqes së implementuar. Arsyeja e zgjedhjes së kësaj forme të provimit të vlerave me JMP të ndërlidhur dhe afër është se, instruksioni CJNE mund të kërcejë relativsht vetëm për +127 dhe -128 bajtë, ndërsa JMP mund kërcejë deri në 32 kB para dhe pas. Rreshti i parë dhe i dytë gjenerohen dhe ekzekutohen vetëm një herë, pasi që në provat tjera përdoret vlera e njëjtë (akumulatori A) tanimë e lexuar nga vendi i caktuar i kujtesës RAM. Ky bllok është

opsional, pasi që nuk gjenerohet kodi i provës së ndërfaqes nëse implementohet vetëm një ndërfaqe.

Blloku 8 poashtu është opsional që përbëhet nga blloqe tjerë dhe gjenerohet nëse ndërfaqja përmban operacione ose atribute.

Blloku 9 është opsional dhe gjenerohet në rastin kur brenda ndërfaqes ka më shumë se një operacion ose atribut dhe që provon nëse është adresuar operacioni (atributi i ekuivalentuar në operacion) i caktuar. Kodi i bllokut 9 është dhënë në shtojcën A.7. Kuptimi dhe roli i bllokut 9 është plotësisht i njëjtë me atë të bllokut 7. Vetëm se tani vlera e ID-së së operacionit gjendet në adresën vijuese 146, në vend se 145 të asaj të ndërfaqes. Rreshtat e kodit në shtojcën A.7 bëjnë provë nëse është adresuar leximi i atributit minTempRead i implementuar në ndërfaqen 0. Kuptimi i shkurtesave të tekstit të labelave është evident. Poashtu brenda bllokut të ndërfaqes kodi në rreshtat 3 dhe 4 me rradhë gjenerohet vetëm një herë, dhe shërben për të provuar nëse është adresuar operacioni i caktuar.

Blloku 10 në fakt është koment mbi vendin në RAM të parametrave hyrës dhe hyrës/dalës ku vendosen gjatë thirrjes së operacionit, dhe vendet e parametrave dalës, hyrës/dalës dhe atij kthyes ku duhet ti vendosë implementuesi i operacionit pas ekzekutimit të operacionit. Një shembull i bllokut 10 mund të duket si në shtojcën A.8. Aty është dhënë gjenerimi i komentit për operacionin standard readTemp si pjesë e ndërfaqes TempSensor (shih listingun 4.5 për IDL). Nga IDL-ja mund të shihet se operacioni në fjalë e ka një parametër hyrës (devID) të tipit short dhe ai në mikrokontrollor 8051 është i vendosur në vendet e kujtesës nga 147 deri 148. Duhet theksuar se bajti me peshë më të madhe vendoset në adresat më të ulëta (147) dhe ai me peshë më të vogël më të larta (149). Parametrin kthyes (return) të operacionit, që është i tipit float, duhet ta vendosim në vendet 144 deri 147 të kujtesës RAM të mikrokontrollorit.

Blloku 11 (shtojca A.9) është poashtu koment dhe opsional pasi që gjenerohet vetëm nëse operacioni standard mund të sinjalizojë gabim dhe në koment qëndrojnë instruksionet se ç'farë duhet të bëjmë nëse operacioni përfundon me sukses ose sinjalizon gabim. Kështu nëse kodi përfundon me sukses atëherë nga komenti kuptohet se duhet të vendoset vlera 0 në akumulator A dhe instruksioni i ardhshëm

duhet të jetë kërcim në labelën `DcdRspns_IntImpl_TempSensor_0_STOP_readTemp`. Në rast se operacioni përfundon me gabimin DeviceNotFound atëherë bajtët e parametrit të vetëm (pasi që mund të ketë më shumë parametra) të gabimit duhet të vendosen në vendet e kujtesës prej 144 deri në 145 dhe pastaj duhet në dekomentohet kodi i dy rreshtave të fundit (MOV A, #1 dhe `JMP DcdRspns_IntImpl_TempSensor_0_STOP_readTemp`).

Blloku opsional 12 gjenerohet në rastin kur kërkohet nga përdoruesi, dhe ka të bëjë me gjenerimin e kodit të leximit dhe vendosjes së vlerës së atributit. Ky kod është i optimizuar ashtu që në bazë të madhësisë së atributit (në bajtë), zgjedh formën adekuate minimale të kodit, për ta kryer funksionin e leximit/vendosjes së atributit. Shembull i kodit të tillë mund të jetë ai në shtojcën A.10, për lexim të atributit minTempRead në ndërfaqen TempSensor.

Në rreshtin 1 regjistri R0 inicializohet me vlerën (128 decimale) e vendit të parë në RAM ku ruhet atributi. Në rreshtin 2 inicializohet regjistri R1 me vlerën e vendit të parë në RAM ku duhet të kopjohet atributi para se të dërgohet. Regjistri R2 inicializohet me vlerën 4 që paraqet (madhësinë) numrin e bajtëve që do të kopjohen. Në rreshtin 4 është labela që përdoret për ciklin e kopjimit të bajtëve. Pasi që nuk mund të kopjohet në menyrë direkte vlera nga vendi R0 në R1, për këtë përdoret akumulatori A. Kështu në rreshtin 5 përmes adresimit indirekt kopjohet vlera nga vendi i treguar me regjistrin R0 në A. Në rreshtin 6, kopjohet vlera A në vendin e treguar me R1. Në rreshtat 7 dhe 8 rriten për një vlerat e regjistrave R0 dhe R1 respektivisht. Me këtë kalohet në vendin vijues të kujtesës. Në rreshtin 9 zvogëlohet vlera e regjistrit R2 (numruesit) për 1 dhe krahasohet nëse kjo vlerë është e ndryshme nga 0. Nëse është e ndryshme nga zeroja, e që praktikisht do të thotë se nuk janë kopjuar të gjithë bajtët e atributit të tipit të dhënë, vazhdohet në instruksionin e treguar me labelën `IntImpl_TempSensor_0_ATTR__get_minTempRead_bytes_loop` në rreshtin 4. Ky cikël përsëritet derisa të kopjohen të gjithë bajtët, pra derisa sa R2 të bëhet 0.

Në bllokun 13 i kthehet përgjigjja klientit duke i thirrur nënprogramet përkatëse. Nëse operacioni sinjalizon gabim(e), dhe kodi është organizuar ashtu siç sugjerohet në koment në bllokun 11 (shtojca A.9), atëherë kodi i bllokut 13 mund të duket si në shtojcën A.11.

Rreshti 1 përmban labelën për provë të statuteve të operacionit, pra paraqet hyrje në kod për provë që vihet si kërcim pas vendosjes së vlerës përkatëse të akumulatorit A, vlerë që përmban statutin e operacionit. Në rreshtin 2 është paraqitur labela që tregon hyrje në kodin që provon nëse operacioni ka përfunduar me status 0 (me sukses). Kështu, rreshti 3 provon nëse vlera e akumulatorit është e ndryshme nga 0. Nëse kjo është e saktë atëherë kërcehet në adresën e treguar me labelën `DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0_Faile d`. Nëse është e pasaktë, pra A është 0, atëherë vazhdohet me ekzekutim në rreshtin vijues 4. Në rreshtin 4 kërcehet në labelën `DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0_RSPNS` ku paraqet fillimin e kodit për kthim të përgjigjes. Rreshti 5 paraqet labelën ku kërcehet nëse A nuk është 0, dhe kjo tregon në instruksionin në rreshtin 6, që është kërcim në labelën `DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_1` që paraqet fillimin e kodit për provë nëse kemi të bëjmë me statutin 1 të operacionit, pra A e barabartë me 1. Në rreshtin 7, siç tregon edhe labela, fillon kodi i kthimit të përgjigjes në rastin kur operacioni kryhet me statut 0. Rreshti 8 e përmban komentin mbi këtë statut. Rreshti 9 vendos vlerën decimale 4 për regjistrin R0, që ka për t'u perdorur për dërgim të 4 bajtëve të *header*-it të përgjigjes. Në rreshtin 10, në regjistrin DPTR vendoset adresa e kujtesës programuese të bajtëve të *head*er-it për statutin 0, përmbajtja e së cilës mund të shihet në bllokun 17. Në rreshtin 11 thirret kodi i nënprogramit për dërgim të bajtëve nga kujtesa programuese e lartpërmendur. Kodi i këtij nënprogrami mund të shihet në bllokun 15. Rreshti 12 paraqet komentin për kodin që pason, pra për dërgim të trupit të mesazhit apo parametrave të operacionit. Në rreshtin 13 vendoset vlera 4 në regjistrin R1, që paraqet madhësinë e të gjithë parametrave kthyes nga operacioni (inout, out dhe return). Ekzekutimi me sukses i operacionit në fjalë (`readTemp`), ka për pasojë kthimin e vetëm një vlere (return)

edhe atë të tipit float, prandaj numri total i bajtëve që kthehen është i barabartë me 4 (vlera e R1). Në rreshtin 14 thirret nënprogrami për dërgim të bajtëve të trupit nga RAM-i. Ky nënprogram është dhënë në bllokun 16. Në rreshtin 15 kërcehet në labelën EnSerComAndInt, që është edhe fundi i kryerjes së operacionit. Përmbajtja e kodit që është pas kësaj labele është paraqitur në bllokun 14 në vijim. Duhet theksuar se, kodi që pason pas labelës `DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_1` (që kërcehet nga rreshti 6) është plotësisht i ngjashëm me kodin e këtij listingu, vetëm se aty referohet në përmbajtje tjetër të *head*er-it dhe trupit të mesazhit. Pra, bëhet fjalë për mesazh tjetër, mesazh me status të ndryshëm prej zeros që paraqet kodin e gabimit. Me bllokun 13 përfundon edhe blloku i gjenerimit të kodit specifik për ndërfaqe dhe operacionet (atributet dhe ata njëkahorë) që i përkasin asaj ndërfaqe.

Blloku 14 paraqet bllokun e kthimit pas ekzekutimit të operacionit në gjendje të pritjes. Ky bllok është dhënë në shtojcën A.12, dhe është fare i shkurtë dhe i thjeshtë.

Rreshti 1 paraqet labelën për hyrje në këtë bllok, që kërcehet nga blloqet tjerë. Rreshti 2 paraqet aktivizimin e mundësisë së pranimit/leximit të bajtëve (kërkesës) përmes portës seriale. Të rikujtojmë, se me rastin e pranimit të tërë kërkesës ishte pamundësuar pranimi i bajtëve tjerë. Në rreshtin 3 aktivizohet gjenerimi i interaptit të portës seriale me rastin e pranimit të bajtëve përmes portës në fjalë. Të rikujtojmë, se me rastin e pranimit të bajtit të parë të kërkesës, ky interapt ishte deaktivizuar për të mos u kthyer në adresë të njëjtë, dhe hyrje në interapt të ri, siç përkon me mënyrën e funksionimit të mikrokontrollorit 8051. Në këtë mënyrë, pas pranimit të bajtit të parë ishte vazhduar me pranim të bajtëve të tjerë (pa interapt) dhe më pastaj përpunim të tyre. Rreshti 4 thjesht i thotë mikrokontrollorit të del nga interapti që është hyrë kur është pranuar bajti i parë. Kështu ekzekutimit të operacionit i përket hyrja në vetëm një interapt, siç e theksuam, duke deaktivizuar atë për bajtët tjerë të pranuar. Dalja nga operacioni bën edhe daljen nga interapti në fjalë.

Blloku 15 shërben për kthim të *header*-it të përgjigjes nga mikrokontrollori. Është i realizuar si nënprogram. Kodi i bllokut 15 është dhënë në shtojcën A.13.

Mund të duket se rreshtat 1 dhe 2 paraqesin redundancë në të dhëna, edhe pse kjo nuk ndikon në madhësi të kodit ekzekutiv, mirëpo kjo e rrit lexueshmërinë e kodit. Në rreshtin 1 arrihet nga blloku 13, ku thirret nënprogrami përmes instruksionit `CALL SendResponseHeader` (shtojca A.11, rreshti 11). Ndërsa në rreshtin 2 arrihet nga instruksioni në rreshtin 9, që do të bëhet fjalë në vijim. Rreshti 3 paraqet instruksionin në të cilin tregojnë labelat në rreshtat 1 dhe 2. Në rreshtin 3 bëhet pastrimi i akumulatorit A ose vendosja e vlerës 0 atij. Kjo për arsye se, instruksioni vijues në rreshtin 4 ia ndan akumulatorit A vlerën nga vendi në kujtesën programuese të treguar nga treguesi (ang. pointer) DPTR plus vlera aktuale e akumulatorit A. Po të ishte A e ndryshme prej 0, instruksioni do ta lexonte vlerën në vendin e kujtesës programuese të zhvendosur për vlerën e akumulatorit A. Me fjalë të tjera A do të paraqiste offset, që është e padëshiruar. Pra, në A vendoset vlera e lexuar nga kujtesa e kodit (programuese) e treguar nga DPTR. Të përkujtojmë se vlera e DPTR caktohet në bllokun 13 (shtojca A.11, rreshti 10). Më tej, përmes instruksionit në rreshtin 5, vlera e lexuar nga vendi i kujtesës programuese përmes akumulatorit A vendoset në buferin serial (SBUF) për dërgim. Në rreshtin 6, pritet derisa të dërgohet bajti në fjalë, që shprehur në kod, pritet dërisa të vendoset vlera 1 në regjistrin TI. Bëhet ekzekutimi i të njëjtit instruksion derisa kjo vlerë të bëhet 1, pra derisa të dërgohet bajti. Pasi të jetë dërguar bajti, vazhdohet me ekzekutimin e istruksionit vijues. Instruksioni vijues (rreshti 7) e bën pastrimin, ose vendosjet e vlerës 0 të bit-regjistrit TI, që e bën të gatshëm për dërgim të bajtit tjetër. Instruksioni vijues në rreshtin 8 e bën rritjen për 1 të vlerës së treguesit DPTR, ose shprehur në fjalë ky tani tregon në bajtin vijues të kujtesës programuese. Në rreshtin 9, zvogëlohet për 1 vlera e regjistrit R0, e inicializuar në bllokun 13 (shtojca A.11, rreshti 9) e që paraqet numrin e bajtëve që përmban *header*-i, dhe shikohet nëse kjo vlerë është e ndryshme prej 0. Nëse është e ndryshme prej 0, që do të thotë kanë mbetur akoma bajtë të *header*-it për t'u dërguar, atëherë kërcëhet në labelën SndByteRH (rreshti 2). Kështu përsëritet me dërgimin e të gjithë bajtëve të *header*-it, pra derisa vlera e R0 të behët 0, që lejon vazhdimin në instruksionin e rradhës. Instruksioni i rradhës dhe i fundit është në rreshtin 10 dhe ky mundëson kthimin nga ky bllok kodi i organizuar si nënprogram, duke vazhduar me ekzekutimin e kodit prej aty nga është thirrur ky nënprogram.

Blloku 16 është i ngjashëm për nga funksionaliteti, por edhe nga dukja, me bllokun 15 të paraqitur paraprakisht. Vetëm se këtu bëhet dërgimi i pjesës së trupit të mesazhit nga kujtesa RAM në vend të asaj programuese. Siç e kemi përmendur edhe në pseudokod, kjo pjesë përbëhet nga parametrat dalës/kthyes të operacionit, më saktësisht ata `inout`, `out` dhe `return`. Kodi i këtij blloku, i organizuar si nënprogram, është dhënë në shtojcën A.14.

Rreshti 1 paraqet labelën përmes së cilës hyhet në këtë nënprogram. Të rikujtojmë se kjo labelë (nënprogram) thirret nga Blloku 13, shtojca A.11 rreshti 14, duke e inicializuar paraprakisht regjistrin R1 në rreshtin 13 (po aty në shtojcën A.11) me numrin e bajtëve që kanë për t'u dërguar nga kujtesa RAM, të vendosur nga përdoruesi në implementim të operacionit. Rreshti 2 (shtojca A.14) e inicializon regjistrin R0 me vlerën (144 decimale) e vendit në kujtesën RAM të bajtit të parë të trupit të mesazhit. Rreshti 3 paraqet labelën në të cilën vihet nga rreshti 9 për secilin bajt që ka për t'u dërguar. Pra kjo labelë dhe rreshti 9 paraqesin ciklin që përsëritet për çdo bajt i trupit të mesazhit nga RAM-i. Në rreshtin 4, përmes adresimit indirekt, nga vendi me adresën e treguar me regjistrin R0 bartet vlera nga ai vend në akumulatorin A. Më tej, në rreshtin 5, kjo vlerë nga akumulatori shkruhet në regjistrin e portës seriale SBUF për dërgim. Në rreshtat 6 dhe 7, njëjtë si te shtojca A.13, bëhet dërgimi i bajtit nga SBUF dhe përgatitet porta për bajtin vijues. Në rreshtin 8 bëhet rritja e vlerës së regjistrit R0 për 1. Tani ky regjistër tregon në adresën e rradhës në RAM. Në rreshtin 9 zvogëlohet për 1 vlera e regjistrit R1, pas dërgimit të çdo bajti, dhe shikohet nëse kjo vlerë është 0. Nëse kjo vlerë është e ndryshme nga 0, që do të thotë se ka akoma bajtë për t'u dërguar, kërcehet në rreshtin 3 për dërgim të bajtit vijues, adresa e të cilit në RAM ndodhet në regjistrin R0. Nëse kjo vlerë është 0, që do të thotë se të gjithe bajtët janë dërguar, vazhdohet me instruksionin në rreshtin 10, ku bëhet kthimi nga nën programi. Kthimi nga nënprogrami ka si pasojë ekzekutimin e kodit në rreshtin vijues pas asaj ku është thirrur ky nënprogram.

Blloku 17 paraqet bazën e bajtëve të header-ëve që ruhen në kujtesën programuese dhe përdoren për dërgim nga blloku 15. Një dukje e mundshme e këtij blloku mund të jetë si në shtojcën A.15.

Labelat para serisë së bajtëve tregojnë mbi *header*-in e përgjigjes së operacionit të caktuar. Kështu rreshti 1 tregon mbi përgjigjen e leximit të atributit `minTempRead` në instancën 0 të ndërfaqes `TempSensor`. Kuptimi i bajtëve varet nga vetë përmbajtja e skedarëve IDL dhe atij implementues. Në rastin konkret, për skedar IDL të dhënë në listingun 4.5 dhe skedar implementues në listingun 5.1, kuptimi i bajtëve të rreshtit 1 është si vijon. Bajti i parë (7), sikur edhe në cilind do rast tjetër edhe në këtë paraqet gjatësinë e mesazhit në bajtë, duke e përfshurë edhe vetë bajtin e parë. Bajti i dytë (0) për këtë rast, paraqet numrin identifikues të ndërfaqes së implementuar. Pasi që implementohen dy ndërfaqe (shih skedarin implementues, listingu 5.1), e ky është i pari në rradhë atëherë ky e ka ID = 0. Bajti i tretë (0) për këtë rast (shih tani skedarin e IDL, listingu 4.5) paraqet numrin identifikues të operacionit. Në ndërfaqen `TempSensor` janë deklaruar dy atribute vetëm të lexueshme (ang. readonly) dhe një operacion standard që sinjalizon gabim. Pasi atributet janë vetëm të lexueshme, për lexim të tyre gjenerohet vetëm nga një operacion (ang. get), pra në total dy. Për operacionin standard gjenerohet vetëm një. Kështu për përcaktim të operacionit të caktuar nevojitet një bajt (pasi numri i operacioneve > 1) dhe ndarja e vlerave fillon nga i pari me vlerën 0. Prej këtu edhe vlera 0 për numrin identifikues të operacionit të leximit të atributit `minTempRead`.

Në anologji mund të shpjegohen p.sh. edhe bajtët e gjeneruar në rreshtin 8. Bajti i parë (6) paraqet gjatësinë e mesazhit. Bajti i dytë (1) paraqet numrin identifikues të ndërfaqes. Vlera 1 vjen pasi skedari implementues përmban dy ndërfaqe dhe kjo ndërfaqe (`Termostat`) është e dyta me rradhë (numrimi fillon nga 0). Bajti i tretë (2) paraqet numrin identifikues të operacionit. Ndërfaqja `Termostat` deklaron tre operacione gjithsejt, dy për lexim të atributeve vetëm të lexueshëm, `minTempAllowed` dhe `maxTempAllowed` respektivisht dhe një për operacionin në fjalë setTemperature. Pasi operacioni është i treti në rradhë, atëherë atij i ndahet ID = 2. Bajti i katërt (1) paraqet statutin e ekzekutimit të operacionit. Siç mund të përfundohet edhe nga labela e rreshtit 8, këtu përcaktohet header-i i mesazhit që kthehet nga serveri i mikrokontrollorit 8051 në rastin kur ndodhë gabimi global `DeviceNotFound` (shih skedarin IDL, listingu 4.5) që ka numrin identifikues 1. Të rikujtojmë se kur operacioni që sinjalizon gabim përfundon me sukses atëherë bajti i

statutit merr vlerë 0. Gabimet globale fillojnë nga vlera 1 deri në 128, ndërsa ato lokale (të deklaruara brenda ndërfaqes) marrin vlera nga 129 deri në 255. Ky gabim (global) sipas përkufizimit, mund të sinjalizohet nga operacionet standarde të cilësdo ndërfaqe.

Kodi i serverit në 8051, i paraqitur përmes blloqeve më herët funksionon si vijon. Ekzekutimi i kodit fillon nga adresa e parë 0000 (blloku 1). Nga këtu kërcehet me ekzekutim në adresën e treguar me labelën InitSerComAndEnblSerInt (blloku 3). Këtu, siç është treguar tanimë bëhet konfigurimi i portës seriale, tajmerit 1 dhe interaptëve përkatës. Më tej vazhdohet në adresën që pason e ajo është labela MAIN (blloku 4). Këtu nuk bëhet asgjë, vetëm pritet për inicim të ekzekutimit të operacioneve me arritjen e bajtit të parë përmes interaptit serial (blloku 5). Në bllokun 5, përveç bajtit të parë lexohen edhe bajtët tjerë të mesazhit të kërkesës. Më tej vazhdohet me ekzekutim të bllokut 6, që përbëhet prej blloqeve të tjerë. Në bllokun 7, që është opsional, shikohet nëse është adresuar ndërfaqja e caktuar. Nëse jo, kërcehet në adresën që krahasohet nëse është adresuar ndërfaqja tjetër (poashtu blloku 7) ose kërcehet në bollokun 14 ku bëhet ri-aktivizimi i portës seriale, me e ç'rast vazhdohet me pritje të kërkesës së re duke e neglizhuar aktualen. Nëse po, pra është adresuar ndërfaqja e caktuar, vazhdohet me ekzekutimin e bllokut 8. Në fillim të bllokut 8 ndodhet blloku 9, ku edhe vazhdohet me ekzekutim nga blloku 7. Në bllokun 9 shikohet nëse është adresuar operacioni i caktuar. Nëse jo, vazhdohet me bllokun tjetër, poashtu 9, ku shikohet nëse është adresuar operacion tjetër (brenda asaj ndërfaqe), ose nëse ska më adresa (operacione) për krahasim kërcehet në bllokun 14 ku më pastaj pritet për kërkesën e rradhës ngjashëm si te rasti i ndërfaqes. Nëse është adresuar operacioni i identifikuar me adresën e dhënë, ekzekutohet kodi i trupit të operacionit në fjalë. Pas kësaj thirret kodi i nënprogrameve të *header*-it (blloku 15) dhe parametrave (blloku 16) të mesazhit. Më pastaj aktivizohet porta seriale duke e thirrur bllokun 14, dhe behët kthimi nga interapti duke vazhduar me pritje për kërkesa të reja, duke ekzekutuar bllokun 4. Duhet theksuar se blloku 4, përveç silljes në vend, mund të përmbajë ç'farëdo kodi që duhet ekzekutuar në pjesën tjetër të kohës, derisa nuk ka kërkesë për ekzekutim të operacioneve.

5.2.2 Struktura e serverit lidhës për JacORB

Kodi i serverit lidhës gjenerohet nga një skedar për secilën ndërfaqe të deklaruar në skedarin implementues të serverit. Para se ndërfaqet të deklarohen në skedarin implementues, ato përcaktohen në skedarin IDL që përshkruan strukturën e ndërfaqeve. Numri i ndërfaqeve në skedarin implementues është vendimtar për gjenerimin e fushës (bajtit) së numrit identifikues në protokollin adaptiv të shkëmbimit të mesazheve. Kështu, për skedarin implementues të dhënë në listingun 5.1, do të gjenerohen dy skedarë të serverëve lidhës: ai për ndërfaqen e parë të implementuar `TempSensor` (me numër identifikues 0), dhe për ndërfaqen e dytë të implementuar `Termostat` (me numër identifikues 1). Skedarin e serverit lidhës të gjeneruar në Java si gjuhë e lartë programuese, po e paraqesim në formë të pseudokodit. Në vend të tërë skedarit, të cilin të pjesshëm e gjejmë në shtojcën B.1, pasi që kodi për secilin operacion është i ngjashëm, këtu do të paraqesim vetëm blloqe të komentuara të kodit të skedarit, të mjaftueshme për kuptim të funksionalitetit. Poashtu, në vend të pjesëve të kodit të të dy skedarëve (ndërfaqeve), do ta paraqesim vetëm pjesën e kodit të skedarit të ndërfaqes `Termostat`. Kodi i skedarit të serverit lidhës të ndërfaqes `TempSensor` është i ngjashëm me atë të ndërfaqes `Termostat`. Në vijim, në listingun 5.2, po e paraqesim pseudokodin e serverit lidhës të realizuar përmes ndërfaqes `Termostat`.

```
class TermostatImpl1 extends TermostatPOA

{
   ... (1) Deklarimi i variablave të lidhjes dhe ndërfaqes

   ... (2) Konstruktorët

   ... (3) Operacionet tjera

    public synchronized int minTempAllowed()
    {
       ... (4) Deklarimi i variablave të operacionit dhe stream-ëve

       ... (5) Lidhja

       ... (6) Shkruarja e parametrave në stream, dërgimi i stream-it
dhe pritja për përgjigje

       ... (7) Leximi dhe analizimi i stream-it përgjigje

       ... (8) Kthimi nga operacioni (me sukses ose gabim)
    }
```

```
}
```

Listingu 5.2 Pseudokodi i serverit lidhës për JacORB i gjeneruar nga ndërfaqja Termostat

Për t'i kuptuar komentet në vijim të blloqeve të pseudokodit, duhet referuar shtojcës B.1 që paraqet pjesën e kodi të serverit lidhës të implementimit të ndërfaqes `Termostat`.

Në fillim gjenerohet emërtimi i klasës së serverit lidhës, `TermostatImpl1`, dhe nga emërtimi mund të shihet se ajo ka të bëjë me ndërfaqen `Termostat`. Pjesa `Impl` tregon se ajo e implementon ndërfaqen në fjalë, ndërsa numri 1 tregon numrin identifikues si ndërfaqe ose server ID gjatë shkëmbimit të mesazheve.

Në bllokun 1 të deklarimit të variableve, gjejmë të gjeneruar variablën `_ms` të tipit `MISPSerialManaged`, që është përgjegjëse për shkëmbim të mesazheve përmes portës seriale. Për klasën `MISPSerialManaged` do të bëhet fjalë në nënkapitullin 5.2.4. Më tej, gjenerohet variabla `_interfaceID` e cila është opsionale, pra gjenerohet vetëm kur në skedarin e implementimit janë dy ose më shumë ndërfaqe. Në këtë rast `_interfaceID` ka vlerën 1, që tregon se është e dyta me rradhë në implementim (pas asaj me ID 0). Pastaj, gjenerohet variabla gjithmonë prezente `_connPolicy`, e cila merr vlerën e nënkuptuar, e në këtë rast `ConnectionPolicy.CONNECTONDEMAND` që ka kuptimin e lidhjes me serverin në mikrokontrollor sipas nevojës. Në rastin kur në skedarin implementues figuron vetëm një ndërfaqe, variabla _connPolicy mërr velërën `ConnectionPolicy.ALWAYSCONNECTED`, që do të thotë rri gjithmonë i lidhur me serverin në mikrokontrollor. Kjo është e logjikshme, pasi që kur në serverin në mikrokontrollor e kemi të implementuar vetëm një ndërfaqe, e kemi edhe vetëm një server lidhës i cili mund të jetë tërë kohën i lidhur me serverin përkatës në mikrokontrollor. Kuptohet, edhe në këtë rast mund të zgjedhet që serveri lidhës të lidhet sipas nevojës, kur ka për të dërguar kërkesë, respektivisht ekzekutuar operacion, por kjo ndikon në performansa (vonesa) për shkak të kohës së nevojshme për lidhje/shkëputje gjatë ekzekutimit të operacionit. Kur i kemi më shumë se dy ndërfaqe të implementuara në mikrokontrollor, të cilave u qasemi përmes serverëve lidhës përkatës, është e nevojshme që të zgjidhet politika e lidhjes

CONNECTONDEMAND. Kjo pasi secili server lidhës përdor instancë të veçantë klase të portës seriale për komunikim, dhe kështu është e nevojshme që pas ekzekutimit të operacionit të inicuar përmes serverit lidhës, të themi me ID 0, të i lejohet mundësia e lidhjes edhe serverit tjetër lidhës, me ID 1, i cili i qaset ndërfaqes përkatëse të impementuar në të njëjtin mikrokontrollor sikur edhe ndërfaqja tjetër (ajo me ID 0). Serveri tjetër mund edhe të ndodhet edhe në tjetër mikrokontrollor, por, nëse të dy mikrokontrollorët janë të lidhur për kompjuter përmes të njëjtës porte seriale, atëherë, serveri lidhës duhet ta lirojë lidhjen pas ekzekutimit të operacionit.

Në bllokun 2, është fshehur përcaktimi i konstruktorëve të serverit lidhës pa parametra dhe me parametra. Konstruktori pa parametra instancon komunikuesin me portën seriale (variablën _ms) me vlera të nënkuptuara të portës seriale, edhe atë me emërtim të portës (ang. port name) 'COM8' dhe shpejtësi shkëmbimi (ang. baud rate) me 1200 bit/s. Për politikë komunikimi merret vlera e parashikuar nga gjeneratori i kodit, tanimë e prezantuar në bllokun 1. Kuptimi dhe roli i parametrave të konstruktorit me parametra është i njëjtë me ata të konstruktorit pa parametra. Në këtë rast kur krijohet (instancohet) serveri lidhës me këtë konstruktor i caktohen parametrat punues.

Edhe pse leximi/vendosja e vlerave të atributit realizohen si operacione të rëndomta, ne në listingun 5.2 e kemi paraqitur në formë të blloqeve pseudokodin e operacionit të leximit të atributit, që do ta analizojmë tani, si dhe në fund dallimet me operacionin e vendosjes së temperaturës. Operacioni i leximit të atributit minTempAllowed (shih listingun 4.5, dhe atributin brenda ndërfaqes Termostat) bëhet përmes metodës përkatëse të përcaktuar në listingun 5.2. Në fillim përcaktohet emërtimi minTempAllowed, parametrat dhe atributet e metodës. Metoda duhet të jetë publike ashtu që t'i qasemi nga klasat tjera përmes mekanizmave të JacORB-it. Poashtu ajo duhet të përmbajë atributin synchronized, që të mund në të njëjtën kohë të inicohet ekzekutimi i vetëm një metode. Kjo, pasi që kodi i mikrokontrollorit është i projektuar ashtu që në të njëjtën kohë të ekzekutohet vetëm një operacion, që i korrespondon metodës përkatëse inicuese në serverin lidhës. Metoda kthen vlerë të tipit int që i korrespondon ekuivalentit long në CORBA, që është vlera e lexuar e atributit. Nuk ka asnjë parametër.

Trupi i metodës fillon me bllokun 4. Në këtë bllok (4) përcaktohet variabla që kthen metoda, e cila lexohet nga mikrokontrollori. Vlera fillestare e kësaj variable është 0. Duhet të inicohet me vlerë fillestare, pasi që në rastin kur dështon komunikimi me serverin në mikrokontrollor, metoda duhet të kthejë vlerë të nënkuptuar që të dilet nga ajo (në bllokun 8). Më tej, përcaktohet numri identifikues (ID) i këtij operacioni që është 0. Pastaj, përcaktohet madhësia e mesazhit që i dërgohet (`_osSize`) dhe atij që pritet të pranohet (`_isSize`) nga mikrokontrollori, respektivisht. Është evidente madhësia 3 e mesazhit që dërgohet, pasi ai përmban bajtët për madhësi të mesazhit, ID të serverit dhe ID të operacionit. Kurse vlera 7 për madhësi të mesazhit që pranohet vie nga 3 bajtët e posa përmendur plus madhësia prej 4 bajtëve e atributit që kthehet (lexohet) nga mikrokontrollori.

Në bllokun 5 bëhet lidhja me portën seriale, parametrat e të cilit janë paracaktuar në konstruktor. Fillimisht provohet nëse serverli lidhës është i shkëputur nga porta, pra, ka kuptim të lidhet vetëm nëse është i shkëputur. Nëse është i lidhur vazhdohet më tej me ekzekutim. Nëse është i shkëputur, provohet të lidhet. Detajet e metodës connect do të jepen në nënkapitullin 5.2.4. Këtu duhet theksuar, se tentohet të lidhet deri në 5 herë, numër ky që mund të ndryshohet. Nëse lidhet vazhdohet me ekzekutim më tej. Nëse nuk lidhet, vazhdohet me ekzekutim, ku paraqitet mesazhi korrespondues ku thuhet se lidhja ka dështuar. Pastaj ekzekutohet kodi më të cilin dilet nga metoda, me ç'rast ajo kthen vlerën e inicuar (në këtë rast 0).

Në fillim të bllokut 6, në rreshtin përkatës instancohet *stream*-i i mesazhit shkues respektivisht kërkesës me madhësi të parallogaritur të ruajtur në variablën `_osSize`. Me rastin e instancimit, në konstruktor, përveç krijimit të *stream*-it me madhësi `_osSize` edhe bajtit të parë të *stream*-it i vëhet vlera `_osSize` konform protokollit AMISP. Për këtë dhe më tepër do të diskutohet në nënkapitullin 5.2.4 ku do te jepen detaje të implemenimit të klasës `StreamOperations2`. Në rreshtat vijues "shkruhen" vlerat e bajtëve të ID të ndërfaqes dhe ID të operacionit në *stream*-in e krijuar në rreshtin në fillim. Vlen të theksohet se me rastin e "shkrimit" në *stream* aplikohen operacionet e ndarjes së tipit shumëbajtësh në bajtë të veçantë, i njohur poashtu edhe si operacion i serializimit. Kjo nuk aplikohet kur vlera që shkruhet në *stream* është e tipit bajt (byte), pasi ajo tanimë është e "transformuar". Është vetë bajti

që kërkohet. Për këto operacione dhe të tjera që i përkasin *stream*-it do të flitet në nënkapitullin 5.4.2. Në rreshtin përkatës vendoset vlera e variablës `isDataReady` të instancës së portës seriale _ms në vlerën false. Kjo bëhet para se të dërgohet *stream*-i i mesazhit kërkesë (shkues) dhe ka rolin të identifikojë se *stream*-i pranues (i përgjigjes) nga mikrokontrollori akoma nuk ka arritur. Variabla `isDataReady` tregon se të dhënat hyrëse akoma nuk janë gati, variabël kjo që caktohet në true nga metodat përkatëse të klasës së portës seriale kur pranohet komplet *stream*-i hyrës (kthyes), pasi që është dërguar ai dalës (shkues). Në rreshtin vijues dërgohet *stream*-i i kërkesës në *stream*-in dalës të instancës së porit serial. Kuptimi i parametrave të metodës write është si vijon. Parametri i parë, `_outStream.getStream()`, paraqet vetë *stream*-in që dërgohet. Parametri i dytë, 0, paraqet offset-in e indeksit të *stream*-it nga i cili fillon dërgimi i bajtëve. Vlera 0 do të thotë fillo dërgimin nga bajti me index 0, pra nga i pari. Parametri i tretë, `_outStream.getStreamLength()`, tregon sa bajtë të *stream*-it të dërgohen. Më këtë rast, numri i bajtëve që duhet dërguar është sa gjatësia e vetë *stream*-it. Pra, dërgohet i tërë *stream*-i paraprakisht i ndërtuar. Në rreshtin vijues hyhet në ciklin while, derisa vlera `isDataReady` tanimë e shpjeguar e portës seriale bëhet true. Kjo do me thënë se, duhet pritur derisa të pranohet përgjigjja nga mikrokontrollori. Pasi të jetë pranuar komplet *stream*-i i përgjigjes, vazhdohet më ndërtimin e tij.

Kështu, në bllokun 7 në rreshtin përkatës krijohet *stream*-i hyrës `_inStream`, nga ai që është pranuar `_ms.readStream`. Në rreshtat vijues, bëhet leximi i të dhënave të nga *stream*-i me ç'rast i ndahet variablave përkatëse, `_readStreamLength`, `_readInterfaceID`, dhe `_readOperationID`. Nga emërtimi lehtë mund të kuptohet edhe roli. Duhet të theksohet se, gjatë leximit të vlerave nga *stream*-i bëhet ndërtimi i vlerave në tipat përkatës nga seria e bajtëve. Kur bëhet ndërtimi i vlerës bajt, siç është rasti me tre rreshtat e përmendur, nuk kemi të bëjmë me ndërtim, pasi, vlera e lexuar (bajt) dhe ajo cak (bajt) janë të njëjta. Bajti i vetëm i lexuar nga *stream*-i paraqet vetë vlerën e kërkuar. Ndërtimi bëhet kur lexohen tipat tjerë nga *stream*-i, siç është p.sh. leximi i vlerës int në rreshtin vijues. Për ndërtimin e vlerave të tipit përkatës nga *stream*-i, do të bëhet fjalë më gjerësisht në nënkapitullin 5.2.4 ku do të jepen detaje të klasës `StreamOperations2`, ku janë

të implementuar operacionet në fjalë. Në rreshtin vijues, provohet nëse parametrat e lexuara të *stream*-it hyrës janë të njëjta me ato të pritura. Mund të shihet se parametrat që priten janë gjatësia e *stream*-it, ID e ndërfaqes dhe ID e operacionit. Nëse vlerat e parametrave të lexuar janë me ata të pritur, vazhdon ekzekutimi në rreshtin përkatës. Aty lexohet nga *stream*-i (ndërtohet nga bajtët e lexuar me metodën readInt) vlera e atributit `minTempAllowed` që i ndahet variablës `_retVal`, që paraqet njëkohësisht edhe vlerën që kthen ky operacion. Nëse ndonjëri nga parametrat nuk ka vlerën e pritur, shtypet mesazhi që paraqet vlerat e pritura dhe ato të lexuara të parametrave, me ç'rast përdoruesi mund lehtë të kuptojë se cili nga parametrat nuk e ka vlerën e pritur. Më tej, ri-vendoset vlera false e variablës `isDataReady` që tregon se puna lidhur me *stream*-in hyrës ka përfunduar dhe se instanca e portës seriale është e lirë për procedurën vijuese të shkruarje/leximit të *stream*-ëve përkatës. Si masë shtesë, kjo vlerë ri-vendoset në false para dërgimit të *stream*-it kërkesë që përgatit instancën e portës seriale për leximin e *stream*-it hyrës. Në rreshtat vijues bëhet trajtimi i gabimit që mund të ndodhë gjatë ndërtimit të *stream*-it kërkesë, dërgimit, pranimit të *stream*-it hyrës, dhe ndërtimit (leximit) të parametrave hyrës. Duhet theksuar se këtu nuk vendoset në false variabla e statutit të leximit të hyrjes (`_ms.isDataReady`), siç është bërë në dy rreshtat e ndarë paraprak, pasi kemi të bëjmë me gabim dhe faktin se nuk dihet nëse *stream*-i është lexuar i tëri (janë lexuar të gjithë parametrat e tij) apo jo. Në këtë rast, duhet analizuar secili nga gabimet që mund të ndodhin dhe të merret masa përkatëse, gjë që, për arsye thjeshtimi të problemit, dhe rrjedhimisht kodit, nuk është bërë.

Në rreshtin vijues (fillimi i bllokut 8), varësisht nga politika e zgjedhur e lidhjes me portën seriale, bëhet shkëputja dhe kështu lirimi i portës seriale për serverë të tjerë lidhës ose jo. Siç e kemi përmendur, kur kemi të implementuar më shumë se një ndërfaqe në mikrokontrollor zgjedhet politika `CONNECTONDEMAND`. Pra, porta seriale lirohet nëse është zgjedhur politika `CONNECTONDEMAND` ose `ADAPTIVECONNECTIVITY`. Nëse është zgjedhur politika `ALWAYSCONNECTED` lidhja me portën seriale mbetet. Në rreshtin vijues, nëse gjithçka ka shkuar si duhet, kthehet vlera e (operacionit) lexuar e atributit nga serveri në mikrokontrollor dhe përfundohet me operacionin. Kjo vlerë përmes mekanizmave CORBA të realizuara në

JacORB i përcillet klientit, që i manifestohet sikur të ishte thirrur lokalisht. Këtu, bashkë me operacionin edhe përfundon edhe blloku 8.

Kod i ngjashëm gjenerohet edhe për operacionin standard `setTemperature` (shih shtojcën B.1). Dallimi është, se këtu mund të gjenerohen gabime (ang. exceptions) si në rreshtat përkatës. Për këtë arsye nevojitet ri-organizim i kodit, ashtu që pjesa e kodit pas pranimit të *stream*-it hyrës, para përpunimit të po atij *stream*-i, të nxjerret jashtë bllokut `try/catch`. Kjo për arsye se, gjenerimi eventual i gabimit do të menaxhohej nga `catch`-i i përmendur dhe kështu nuk do të i përcillej klientit. Gjenerimi i gabimit duhet të i lihet mekanizmave CORBA. Mund të shihet edhe se rreshtat e lirimit ose jo të portës seriale, nuk është lënë për në fund para daljes nga operacioni për arsye të njëjta. Gjenerimi i gabimit do ta ndërpriste ekzekutimin e mëtujeshëm të kodit, me ç'rast lidhja do të mbetej ashtu si është, e hapur, gjë që nuk është e dëshirueshme. Si provë shtesë, për dallim nga atributi, këtu shikohet nëse vlera e variablës (`_readStatusID`) së statutit është e barabartë me 0. Vlera 0 tregon për përfundimin normal të operacionit. Poashtu edhe vlera e pritur e variablës së gjatësisë (`_isSize`), ka kuptimin e gjatësisë së *stream*-it për rastin e përfundimit pa gabime të operacionit. Për rastet tjera të përfundimit, pra në rastet kur përfundohet me ndonjë gabim, provohet nëse variabla e lexuar e gjatësisë `_readStreamLength` ka vlerën e caktuar. Nëse janë plotësuar kushtet që operacioni është ekzekutuar në mikrokontrollor pa gabime. Në këtë rast operacioni duhet të përfundojë normalisht. Në rreshtin vijues provohet nëse operacioni në fjalë, i identifikuar me parametrat përkatës, përfundon me statutin me vlerë 1. Kjo vlerë paraqet kodit e gabimit të parë dhe të vetëm global (`DeviceNotFound`) të deklaruar në skedarin IDL (shih listingun 4.5). Nëse operacioni i inicuar në mikrokontrollor ka përfunduar me statutin 1, atëherë gjenerohet gabimi përkatës, `DeviceNotFound`, me parametrin e tipit ekuivalent në Java short, që lexohet nga *stream*-i hyrës dhe i përcillet klasës së gabimit në fjalë. Ngjashëm, në rreshtin vijues provohet nëse operacioni në fjalë, në mikrokontrollor ka përfunduar me statut të barabartë me 129. Kjo vlerë paraqet numrin identifikues të gabimit të parë dhe të vetëm lokal `TempNotAllowed` të deklaruar brenda ndërfaqes `Termostat`. Nëse plotësohet kushti i shtruar, atëherë vazhdohet me ekzekutim në rreshtin përkatës, aty ku

gjenerohet gabimi lokal i përmendur. Mund të shihet se, klasa e gabimit lokal `TempNotAllowed` e përcaktuar brenda pakos TermostatPackage, pranon dy argumente hyrëse (`_inStream.readShort()` dhe `_inStream.readFloat()`) që lexohen nga *stream*-i hyrës. Njëri është i tipit short, që paraqet vlerën e parametrit `deviceID`, kurse tjetri i tipit float, që paraqet vlerën e parametrit temperature të deklaruar në listingun 4.5. Nëse asnjëri nga kushtet nuk plotësohet, vazhdohet me bllokun që përmban instruksionin për shtypje të mesazhit të gabimit. Pasi asnjëri nga kombinimi i parametrave të pritur nuk është kthyer, atëherë ky mesazh shtyp vlerat parametrave të lexuar, ashtu që të mund të detektohet parametri i kthyer gabimisht (nga mikrokontrollori).

5.2.3 Struktura e serverit lidhës për Shërbime Ueb

Për dallim nga gjenerimi i kodit për teknologjinë JacORB, për të cilën gjenerohen vetëm serverët lidhës të deklaruar në skedarin implementues, për secilin nga një, për Shërbimet Ueb duhet gjeneruar edhe skedarë tjerë. Kjo për arsye se p.sh. skedarin e ndërfaqes dhe klasat e gabimeve si dhe klasat tjera të nevojshme në JacORB i gjeneron gjeneratori i JacORB, kurse ke Shërbimet Ueb është dashur të zhvillohen edhe gjeneratorët e këtyre skedarëve, të nevojshëm për funksionim normal të serverit lidhës. Klasa implementuese e ndërfaqes, pra ajo e serverit lidhës, siç është edhe theksuar në kapitullin 3 (paragrafi 3.4), për alternativën e Shërbimeve Ueb është thuaja se e njëjtë me atë të JacORB. Dallimet janë në disa rreshta të kodit.

Kështu, për dallim nga ekuivalenti në JacORB në fillim importohen klasat specifike për Shërbimet Ueb në Java. Pastaj, vendoset shënimi (ang. annotation) që përcakton se serveri është me gjendje. Pa këtë shënim, rëndom serveri krijohet pa gjendje. Kjo do të shkaktonte ri-instancimin të klasës së portës seriale, sa herë që inicohet operacion nga ana e klientit, që do të kishte ndikim (dobësim) në performansa. Më tej, caktohen parametrat e Shërbimit Ueb (serverit). Klasa `TermostatImpl1` implementon ndërfaqen `Termostat`, i cili poashtu gjenerohet dhe përmbajtja e të cilit jepet në shtojcën C.1. Rreshtit që paraqet deklarim të operacionit për lexim të atributit, për dallim nga ai në JacORB, emërtimit `minTempAllowed` i është shtuar përpara edhe shkurtesa get_. Kjo për arsye se në

Shërbime Ueb nuk bëhet dallimi si në JacORB përmes *signature* të operacioneve, pra lejimi i operacioneve me emërtim të njëjtë por me parametra të tipave të ndryshëm. Në Shërbime Ueb secili operacion duhet të ketë emërtim unik.

Dallimi tjetër paraqtitet te deklarimi i operacionit `setTemperature`, i cili dallon vetëm te specifikimi i gabimeve që mund të ndodhin. Në vend se të specifikohet se p.sh. mund të sinjalizojë gabimin global `DeviceNotFound`, duhet të figurojë klasa fault `DeviceNotFound_Exception`, e cila përmban në vete si atribut klasën përkatëse `DeviceNotFound`. Të dyja këto klasë duhet dhe gjenerohen nga gjeneratori jonë i kodit, detajet e të cilave do të jepen në vijim. Rrjedhimisht, edhe kodi për sinjalizim të gabimit duhet të bëhet konform klasëve të përmendura të gabimeve. Dallimi është i thjeshtë, në vend se të sinjalizohet direkt gabimi si `throw new DeviceNotFound(_inStream.readShort());`, ai sinjalizohet si `throw new DeviceNotFound_Exception("Global exception (1) raised!", new DeviceNotFound(_inStream.readShort()));`. Parametri i parë i klasës fault paraqet koment, ndërsa i dyti vetë klasën e gabimit. Dallim tjetër që ekziston, e që nuk mund të vërehet me shembullin e paraqitur është gjenerimi i parametrave hyrës/dalës dhe dalës. JacORB përdor tjera klasë (shih tabelën 4.2, kolona 'Klasa Holder'), ndërsa Shërbimet Ueb tjera për secilin nga tipat themelorë të të dhënave në Java. Përveç dallimit në klasë, deklarimi dallon edhe në sintaksë, ku te Shërbimet Ueb përdoren shënimet (ang. annotations) për specifikim të drejtimit të parametrave. Në cilindo nga rastet, leximi dhe caktimi i vlerës së parametrit bëhet në formën `parametri.value`.

Përveç skedarëve/klasëve implementues që gjenerohen për çdo instancë të tyre në skedarin implementues (.impl), duhet të gjenerohen edhe ndërfaqet përkatëse nga skedari IDL (.idl) që implementohen nga skedarët implementues. Kështu, klasa `TermostatImpl1` e paraqitur në shtojcën C.1 implementon ndëraqen `Termostat`, të cilën e kemi paraqitur në shtojcën C.2.

Përveç ndërfaqeve gjenerohen edhe klasat që përmbajnë gabimet (lokalë dhe globalë), si dhe klasat përkatëse *fault* për secilin gabim. Për gabimet lokalë, klasat

gjenerohen në direktori dhe pako, me emër të ndërfaqes të cilës i përkasin dhe duke i shtuar prapashtesën `Package`, për ta ruajtur ngjashmërinë me JacORB. Ato globale, gjenerohen në direktorinë ku gjenerohen ndërfaqet dhe klasat implementuese të ndërfaqeve.

Në shtojcën C.3 është paraqitur kodi i gabimit lokal `TempNotAllowed` të deklaruar brenda ndërfaqes `Termostat`. Ai gjenerohet në direktorinë `TermostatPackage`, dhe është i deklaruar brenda pakos me emër të njëjtë me direktorinë (`TermostatPackage`). Në fillim, siç u përmend, përcaktohet pakoja të cilës i takon, që është e njëjtë me direktorinë në të cilën gjenerohet. Më tej, importohet klasa `Exception`, të cilën e zgjeron klasa e gabimit në fjalë. Pastaj, deklarohen atributet e gabimit me tipa në Java, konform tipave përkatës siç janë deklaruar në skedarin IDL të gabimit në fjalë. Është deklaruar edhe konstruktori, që thirret nga skedari implementues i serverit lidhës.

Në shtojcën C.4 është dhënë skedari i klasës *fault*, i gjeneruar për gabimin `TempNotAllowed`. Përmbajtja e këtij skedari është thuaja statike, duke ndryshuar vetëm pjesët që përcaktojnë gabimin si dhe ndërfaqen brenda së cilës është deklaruar `(targetNamespace = "Termostat")`. Kjo e fundit është opsionale dhe është blanko për gabimet globale `(targetNamespace = "")`.

5.2.4 Klasat ndihmëse MISPSerialManaged dhe StreamOperations2

Klasat `MISPSerialManaged` dhe `StreamOperations2` përdoren nga serveri lidhës për realizim të komunikimit me mikrokontrollor. Përderisa klasa `MISPSerialManaged` përdoret për lidhje dhe shkëmbim të bajtëve (ang. stream) përmes portës seriale, klasa `StreamOperations2` përdoret për ndërtim të *stream*-it, zbërthim dhe lexim të të dhënave të serializuara në *stream*. Emërtimet fillestare të klasëve `MISPSerialManaged` dhe `StreamOperations2` ishin `MISPSerial` dhe `StreamOperations`, respektivisht. Me përkrahjen për lidhje/shkëputje sipas nevojës (i menaxhuar), nga i lidhur tërë kohën, përmbajtja e klasës `MISPSerial` ndryshoi dhe kështu edhe emërtimi në `MISPSerialManaged`. Ngjashëm edhe për shkak të ndryshimeve të vogla në klasën `StreamOperations` është ndërruar

emërtimi në `StreamOperations2`. Në vijim po i paraqesim disa nga detajet e vetëm versioneve të fundit.

5.2.4.1 Klasa MISPSerialManaged

Siç u përmend më parë klasa `MISPSerialManaged` shërben për manipulim me portën seriale. Përdoret nga serveri lidhës për realizim të veprimeve themelore të komunikimit me serverin në mikrokontrollor përmes portës seriale. Veprimet kryesore janë lidhja me portë, dërgimi i kërkesës, pritja për përgjigje dhe shkëputja me portën. Mikrokontrollori lidhet në portën seriale të caktuar dhe pret për kërkesa nga serveri lidhës.

Klasa `MISPSerialManaged` përbëhet prej dy konstruktorëve dhe disa metodave, prej të cilave disa ekzekutohen si *threads*. Konstruktorët bëjnë listimin e porteve seriale të kompjuterit dhe provojnë nëse porta e specifikuar përmes emërtimit si string ekziston. Konstruktorët janë identik, përveç dallimit në argumente: i pari, pa argumente hyrëse, provon për vlera të nënkuputuara të portës (emërtimi i portit: COM8 dhe shpejtësia: 1200 baud/s), ndërsa i dyti, me argumente hyrëse vlerat e dhëna të portës nga përdoruesi.

Lidhja me portën seriale bëhet përmes metodës `connect` e cila kthen vlerë logjike (`boolean`) mbi suksesin (`true`) ose dështimin (`false`) e provës për lidhje me portën e dhënë. Metoda provon numër të caktuar herësh për lidhje me portën seriale duke e thirur metodën tjetër `openPort`. Nëse lidhja ka sukses, vazhdohet më tej ku bëhet vendosja e vlerës së atributit `isConnected` në atë të statutit të lidhjes, që në këtë rast është `true`, dhe dilet nga metoda duke kthyer poashtu vlerën `true`. Nëse lidhja dështon, variabla që numron provat për lidhjet rritet për një dhe pritet për kohë të caktuar (të dhënë në milisekonda) deri në provën e ardhshme. Ky cikël përsëritet, përderisa numri i provave të dështuara është më i vogël se vlera e paracaktuar maksimale e lejuar e tyre. Nëse ky numër i lejuar i provave tejkalohet, atributi `isConnected` si dhe vlera që kthen metoda marrin vlerën logjike `false`, që tregojnë mbi dështimin e lidhjes. Metoda në fjalë është paraqitur në shtojcën D.1.

Dërgimi i bajtëve të kërkesës bëhet përmes metodës `write` të atributit `outputStream`. Metoda `write` pranon si hyrje varg të tipit `byte`.

Pranimi i bajtëve bëhet përmes metodës `serialEvent` e cila në të vërtetë i përpunon disa ngjarje, në mesin e tyre edhe atë të dispozicionit të të dhënave (bajtëve) në portë (`DATA_AVAILABLE`). Pasi shpesh përgjigjja nuk vjen e tëra, por pjesë pjesë në grupe bajtësh, atëherë është e nevojshme që grupet e bajtëve të rradhiten në varg sipas rradhës që kanë ardhur, i pari në fillim kurse i fundit në fund. Bajti i parë i përgjigjes vendoset në varg në pozitën me indeks 0. Siç dihet, vlera e këtij bajti paraqet gjatësinë e *stream*-it. Bajtët tjerë, varësisht nga numri i tyre vendosen në vargun e përgjigjes, dhe me këtë rast kur pranohet bajti i fundit bëhet edhe vënia e vlerës së atributit `isDataReady` në `true`, që tregon se është pranuar e tërë përgjigjja (të gjithë bajtët e saj). Sekuencën e kodit të metodës `serialEvent` që kryen veprimet e posapërmendura, e kemi paraqitur në shtojcën D.2.

Shkëputja nga porta bëhet përnes metodës `disconnect` që e thirr metodën tjetër `closePort`. Sikurse te lidhja, edhe këtu bëhet vënia e vlerës së atributit `isConnected` në `true` nëse metoda `closePort` ekzekutohet me sukses. Në shtojcën D.3 është paraqitur metoda `disconnect`.

Duhet theksuar se klasa implementon edhe metoda tjera ndihmëse përveç këtyre bazike të përmendura. Ato mundësojnë funksionimin e metodave të komentuara dhe në përgjithësi të komunikimit serial.

5.2.4.2 Klasa StreamOperations2

Klasa `StreamOperations2` përdoret për manipulim me *stream*: krijim të tij, shkruarje në *stream* ose serializim të të dhënave (parametrave), dhe lexim ose deserializim të të dhënave. Përbëhet prej dy konstruktorëve. Konstruktori që ka parametër gjatësinë e *stream*-it. përdoret për ndërtim të kërkesës, ndërsa ai që ka parametër seri të bajtëve shërben për lexim të të dhënave të përgjigjes. Në shtojcën D.4 janë dhënë konstruktorët në fjalë. Vlen të përmendet atributi `indxPtr`. Ai paraqet treguesin e indeksit, vlera e të cilit në rastin e parë incializohet në vlerën 1, ndërsa në rastin e dytë në 0. Kjo për arsye se në rastin e parë, që përdoret për ndërtim

(kërkesë), vlera e gjatësisë dihet pasi jepet si parametër dhe bëhet vënia e asaj vlere anëtarit të parë të vargut (`stream[0] = streamSize;`). Në vend se të inkrementohet për një (nga vlera 0), për të arritur performansë thjesht kjo vlerë vëhet si 1. Për rastin e dytë, situata është evidente. Vlera e treguesit të indeksit është 0, që do të thotë se jemi në fillim të *stream*–it dhe asnjë nga të dhënat nuk janë lexuar akoma.

Operacionet me *stream* janë ato të shkruarjes së një vlere të tipit të caktuar në të (serializim), dhe leximi i vlerës së tipit të caktuar nga ai (deserializim). Në shtojcën D.5 janë paraqitur këto dy operacione, i shkruarjes dhe i leximit, respektivisht, për të dhënë të tipit int të Java-s. Siç mund të shihet nga shtojca D.5 kemi përdorim të operacioneve të zhvendosjes së bitëve për pozita të caktuara. Për operacionin e shkrimit (`writeInt`) bajti i caktuar zhvendoset përmes operatorit `>>`, eliminohen bajtët tjerë përmes operacionit `& 0xff`, dhe pastaj kjo vlerë i ndahet anëtarit të caktuar të *stream*-it `stream`, duke e inkrementuar më pastaj vlerën e treguesit të indeksit për një `indxPtr++`. Inkrementimi i treguesit të indeksit për një do të thotë bëhu gati për tregim (shkruarje) në pozitën e ardhshme të *stream*-it.

Raste interesante paraqesin shkrimi dhe leximi i të dhënave të tipave `boolean` dhe me pikë të lëvizshme (`float` dhe `double`), që e dallojnë nga ata numerik të plotë.

Në shtojcën D.6 është paraqitur shkrimi dhe leximi, respektivisht, i të dhënës së tipit booean në *stream*. Nga shtojca D.6 mund të shihet se e dhëna e tipit `boolean` shkruhet në një bajt të vetëm në *stream*-it. Në rastin kur është `true` shkruhet si 1, ndërsa kur është `false` si 0. Ngjashëm vlen edhe për leximin nga *stream*-i: kur bajti që paraqet vlerën `boolean` është i ndryshëm nga 0 kthehet vlera `true`, ndërsa kur është 0 kthehet vlera `false`.

Në shtojcën D.7 është paraqitur shkrimi dhe leximi, respektivisht, i të dhënës së tipit `float` në *stream*. Nga shtojca D.7 mund të kuptohet se te rasti i shkrimit në `stream` (operacioni `writeFloat`) së pari duhet që vlera e tipit `float` të konvertohet në ekuivalentin `int` përmes funksionit `Float.floatToIntBits(fVal)`. Më pastaj mund të vazhdohet zbërthimi në

bajtë të veçantë i vlerës së tipit `int`, sikur në rastin e operacionit `writeInt` listingu D.5. Operacioni i leximit nga *stream*-i në `float` (`readFloat`) është paksa më ndryshe. Së pari ndërtohet nënvargu `b` nga vargu i *stream*-it `stream` dhe pastaj përmes funksionit `ByteBuffer.wrap(b).getFloat()` ky nënvarg kthehet në `float`.

5.3 Gjeneratorët e kodit

Gjeneratorët e kodit paraqesin kontributin kryesor të kësaj teze. Për ndërtimin e tyre është përdorur vegla e specializuar për ndërtim të kompilatorëve, respektivisht gjeneratorëve të kodit Coco/R [1]. Përmes gjeneratorëve gjenerohet serveri në pajisje të mbyllur – mikrokontrollor dhe serverët lidhës. Gjenerohen dy lloje të serverëve lidhës: ata për alternativën në Java të CORBA-s e njohur si JacORB, dhe alternativën në Java të Shërbimeve Ueb e njohur si JAX-WS.

Hapi i parë drejt ndërtimit të gjeneratorit të kodit është shkruarja e gramataikës me aksionet semantike brenda saj. Gramatika përkufizon gjuhën të cilën e njeh gjeneratori. Aksionet semantike paraqesin pjesë kodi në gjuhën implementuese të gjeneratorit, që në rastin tonë është Java, që kryejnë aksione të caktuara kur haset përkufizimi i caktuar me gramatikën përkatëse. Përmes aksioneve semantike ndërtohen listat e objekteve, që paraqesin versionin në kujtesë të kompjuterit të skedarëve IDL dhe atij implementues. Lista e objekteve paraqet klasat përkatëse të entiteteve të caktuar, si: ndërfaqet, gabimet, operacionet dhe atributet, të lidhur në mes vete si lista të lidhura (ang. linked list) ose ndryshe. Listat e objekteve më pastaj përdoren nga gjeneratorët përkatës, për gjenerim të serverëve lidhës dhe atij të mbyllur.

5.3.1 Gramatika, aksionet semantike, dhe klasat e nevojshme për ndërtim të listës së objekteve

Për ndërtim të gjeneratorëve të kodit është dashur përkufizuar dy gramatika: atë të skedarit IDL dhe atij implementues. Gramatika e skedarit implementues është fare e thjeshtë, dhe shfrytëzon listën e objekteve të skedarit IDL për ndërtim të listës së vet të objekteve. Pra, objektet e saj janë objekte të skedarit IDL. Gramatika e

skedarit IDL është dhënë në listingun 3.1, pa aksione semantike. Do të i shohim disa nga rregullat e saj me aksione semantike. Gramatikën e skedarit implementues e kemi paraqitur në shtojcën E.1, me aksione semantike.

Gramatika e skedarit implementues në shtojcën E.1 lejon vetëm deklarimin e ndërfaqeve. Pra, në skedarin implementues mund të deklarohen vetëm ndërfaqe, dhe rrjedhimisht në pajisje të mbyllur (mikrokontrollor) mund të implementohen vetëm ndërfaqe. Ndërfaqet mund të përmbajnë operacione standarde që mund të sinjalizojnë gabime globale dhe lokale, si dhe operacione njëkahëshe dhe atribute, të përcaktuar në skedarin IDL. Me fjalë të tjera, skedari implementues shfrytëzon strukturën me përkufizime të skedarit IDL. Instruksionet (në shtojcën E.1) në rreshtat 16 dhe 25, kthejnë ndërfaqen nga lista e objekteve të skedarit IDL në bazë të emrit të dhënë në skedarin implementues. Kështu, ndërtohet lista e objekteve të skedarit implementues me ndërfaqe nga lista e objekteve (nga ndërfaqet) të skedarit IDL.

Për gramatikën e skedarit IDL, po i paraqesim disa nga rregullat e listingut 3.1 me aksionet semantike. Konkretisht, po e paraqesim futjen e ndërfaqeve në listë të objekteve, si dhe të operacioneve standarde.

Në shtojcën E.2 është ri-paraqitur rregulla me numër identifikues 2 (përcaktimi i ndërfaqes) nga listingu 3.1, pjesa para shenjës së barazimit (=). Por, në këtë rast me aksionet semantike për regjistrim të ndërfaqes në listë të objekteve.

Nga kodi në shtojcën E.2 mund të kuptohet se, nëse lista është boshe atëherë ndërfaqja bëhet elementi i parë i listës (rreshtat 2 deri në 6), ndërsa nëse nuk është boshe regjistrohet si anëtari vijues i listës së ndërlidhur (rreshtat 8 deri në 13). Më tej, në rreshtat 15 deri në 19, caktohet lloji i objektit. Rreshtat 15 deri në 17 e caktojnë objektin e listës ndërfaqe, ndërsa rreshti 19 e reseton numratorin e gabimeve lokale në vlerën fillestare të rezervuar për gabime lokale (129). Kështu, gabimet lokale brenda ndërfaqes së posa futur në listë marrin numra identifikues duke filluar nga vlera e caktuar (129).

Për arsye sqarimi, gramatikën e ndërfaqes po e paraqesim përmes diagramit sintaksor në figurën 5.2.

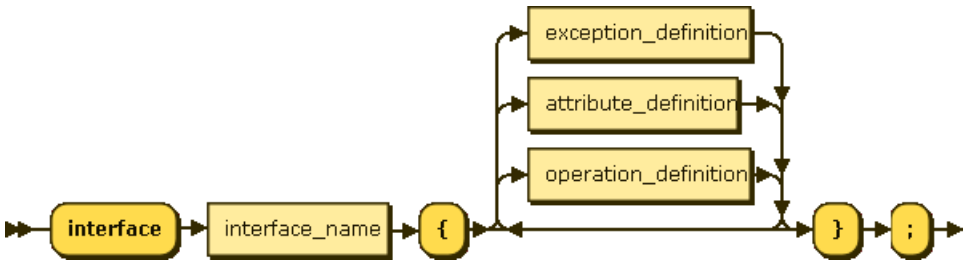


Figura 5.2 Diagrami sintaksor i ndërfaqes

Në shtojcën E.3 është ri-paraqitur rregulla me numër identifikues 7 (përcaktimi i operacionit) nga listingu 3.1, me aksionet semantike për regjistrim të operacionit standard në kuadër të ndërfaqes në listë të objekteve. Në rreshtat 2 deri në 13 bëhet regjistrimi i operacionit standard në kuadër të ndërfaqes (`idle_intrfc`) si element i ndërfaqes në fjalë. Në rreshtat 15 deri në 17 përcaktohet si operacion standard duke përfshirë edhe numrin identifikues të tipit (1). Në rershtat 20 deri në 23 i caktohet parametri që e kthen (ang. return) operacioni. Në rreshtin 25 caktohet emërtimi i operacionit. Në rreshtat 27 deri në 29 regjistrohet gabimi i parë në listë të gabimeve brenda operacionit, që mund ta sinjalizojë operacioni standard në fjalë. Në rreshtat 31 deri në 35 regjistrohen gabimet tjerë në listë të gabimeve brenda të njëjtit operacion standard. Në rreshtin 26 figuron rregulla e transformimit (`methodparammetersbody`), e cila përmban parametrat (argumentet) e operacionit, e që transformohet më tej si në shtojcën E.4.

Nga shtojca E.4, mund të kuptohet se rregulla methodparammetersbody transformohet më tej si në rreshtin 1. Në rreshtin 2 fillon përkufizimi për rregullën `methodparammeter`, që figuron në transformimin në rreshtin 1. Siç tregon edhe emri, `methodparammeter` paraqet përkufizimin e parametrit (argumentit) të operacionit. Në rreshtat 3 deri në 14 bëhet regjistrimi i parametrit në listën e parametrave të operacionit standard. Në rreshtat 16 deri në 18, caktohet drejtimi i parametrit që mund të jetë i llojit `in`, `out` ose `inout`. Në rreshtat 21 deri në 22 caktohet tipi dhe madhësia e parametrit, ndërsa në rreshtin 23 emri i parametrit.

Për arsye sqarimi, gramatikën e operacionit standard po e paraqesim përmes diagramit sintaksor në figurën 5.3.

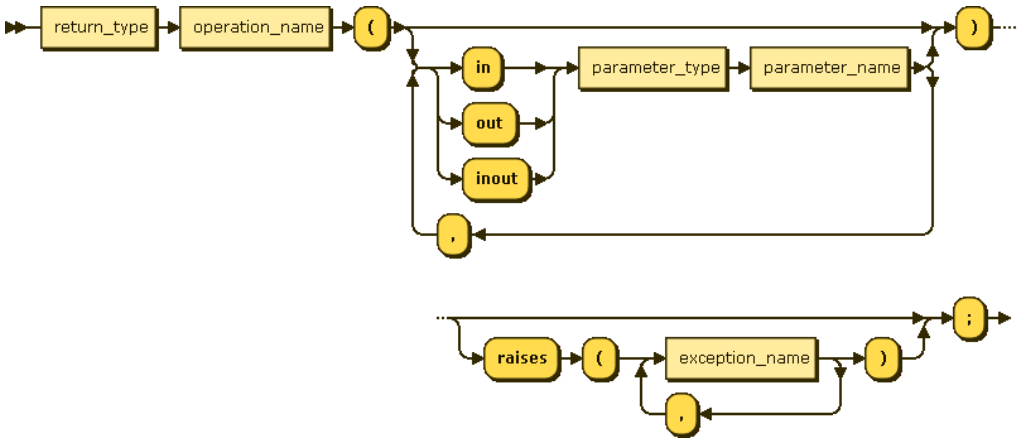


Figura 5.3 Diagrami sintaksor i operacionit standard

Nga shtojcat e paraqitura në këtë seksion mund të vërehet se përdoret klasa `IDLEntities` me funksionet për kthim të vlerave të caktuara për hyrje të dhëna, si dhe klasat për ruajtje të objekteve si ndërfaqe, operacion standard, parametër, gabim, etj. Funksione të klasës `IDLEntities` që figurojnë në shtojcat e përmendura janë p.sh.: funksioni `getInterface` në shtojcën E.1 rreshtat 16 dhe 25. Ky funksion kthen strukturën (instancën e klasës) e ndërfaqes nga lista e objekteve të skedarit IDL, në bazë të emrit të ndërfaqes të dhënë si hyrje. Implementimi i këtij funksioni brenda klasës `IDLEntities` duket si në shtojcën E.5.

Tjerë funksione nga klasa `IDLEntities` janë edhe p.sh. `getException` që paraqitet në shtojcën E.3 rreshtat 29 dhe 35. Ky funksion kthen instancën e klasës së gabimit në bazë të emërtimit të tij. Shtojca E.6 paraqet funksionin `getException`. Në rreshtat 6 deri në 15 shikohet nëse gabimi është pjesë e listës globale të objekteve të skedarit IDL. Pavarësisht gjetjes në listën globale të objekteve, në rreshtat 18 deri në 27 vazhdohet të shikohet nëse është pjesë e listës së objekteve të ndërfaqes së dhënë. Kjo për arsye se nëse një gabim me emër të njëjtë përcaktohet në të dy listat, ajo lokale ka përparësi sepse gabimi mbishkruhet, ngjashëm me variablat e deklaruara në blloqe të ndryshme. Brenda listës globale gabimet përcaktohen me lloj të objektit 2 (rreshti 8), ndërsa brenda listës së objekteve të ndërfaqes përcaktohen me lloj të objektit 4 (rreshti 20).

Gjatë parsimit të skedarit hyrës IDL, në pjesë të caktuara të dokumentit ekzekutohen aksionet semantike përkatëse. Aksionet semantike bëjnë ndërtimin e

listës së objekteve. Lista e objekteve ndërtohet nga klasat e entiteteve: ndërfaqes, gabimit, operacionit standard, operacionit njëkahësh dhe atributit. Klasa e listës globale të objekteve është `IDLElement` e paraqitur në shtojcën E.7. Në rreshtin 4, atributi type shërben për ruajtje të llojit të objektit, ndërsa object paraqet vetë instancën e objektit. Për gramatikën e dhënë aktuale të skedarit IDL (listingu 3.1), objekti mund të jetë ndërfaqe (type = 1) ose gabim (type = 2). Atributet `first`, `previous` dhe `next` paraqesin respektivisht entitetin (elementin) e parë, paraprak dhe atë të ardhshëm të listës në krahasim me objektin aktual.

Objekt tipik i klasës `IDLElement` është ndërfaqja, e organizuar (ruajtur) me klasën `IDLEInterface` të paraqitur në shtojcën E.8.

Kuptimi i atributeve në klasën `IDLEInterface` është si vijon. Atributi `interfaceName` është për emërtimin e ndërfaqes. `isSEASecured` është i rezervuar për komunikim të sigurt (enkriptuar) ndërmjet serverit lidhës (implementimit të ndërfaqes) dhe serverit ne mikrokontrollor. Atributi `generateOperationID` shërben për të treguar nëse duhet të gjenerohet kodi për provë nëse është adresuar operacioni i caktuar. Vlera e këtij atributi caktohet gjatë pre-kalkulimeve, para gjenerimit të kodit, pasi të jetë ndërtuar në tërësi ndërfaqja. Atributet `firstie` dhe `currentie` paraqesin elementin e parë dhe atë aktual, respektivisht, të listës së elementeve përbërës të ndërfaqes. Elementet përbërëse të ndërfaqes mund të jenë gabimet, operacionet standarde, operacionet njëkahore dhe atributet. Përkufizimi i klasës `InterfaceElement` të elementeve të ndërfaqes është paraqitur në shtojcën E.9.

Kuptimi i atributeve në klasën `InterfaceElement` është i ngjashëm me ato të klasës `IDLElement` tanimë të paraqitur. Pasi klasa është e organizuar si listë e lidhur dyfish, atributet previous dhe next paraqesin elementin paraprak dhe atë vijues të listës, respektivisht. Atributi type përmban llojin e objektit të elementit. Për secilin nga llojet, gabim, operacion standard, operacion njëkahor dhe atribut, është caktuar nga një numër identifikues. Atributi object paraqet vet objektin e llojit të përmendur. Për rastin e operacionit standard, klasa që përfaqëson atë është `IEStandardOperation`, e paraqitur në shtojcën E.10.

Edhe klasa `IEStandardOperation` i ka disa atribute që e karakterizojnë operacionin standard. Kështu, atributi `retParam` i tipit `Parameter` shërben për ruatje të vlerës që kthen (`return`) operacioni. Krahas tipave themelorë, ky atribut mund të jetë edhe `void`, në rastet kur operacioni nuk kthen vlerë. Atributi `operationID` shërben për ruatje të emërtimit (identifikues) të operacionit. Atributet `firstParam` dhe `params` shërbejnë për ruajtje të parametrit (argumentit) të parë, respektivisht atij të fundit, të operacionit. Atributet `firstExcptn` dhe `excptn` shërbejnë për ruajtje të gabimit të parë dhe atij të fundit, respektivisht, që mund të i sinjalizojë operacioni, sipas rradhës që paraqiten gjatë përcaktimit të operacionit në skedarin IDL. Vlera e atributit `generateStatusID` caktohet gjatë parallogaritjeve, para gjenerimit të kodit, dhe kjo vlerë tregon nëse duhet të gjenerohet fusha (bajti) e statutit të ekzekutimit të operacionit. Në rastet kur operacioni nuk sinjalizon gabime, kjo vlerë është false, pra, nuk gjenerohet bajti i statutit. Nga shtojca E.10 mund të kuptohet se përdoren klasat `Parameter` dhe `IDLEExceptionList`. Përkufizimi i klasës `Parameter` është dhënë në shtojcën E.11.

Siç mund të shihet nga shtojca E.11 klasa `Parameter` paraqet listë të ndërlidhur, ku atributet `previous` dhe `next` paraqesin elementin paraprak dhe atë vijues, respektivisht. Atributi `direction` përmban informatën për drejtimin e parametrit (argumentit). Nga shtojca E.4 mund të kuptohet se për parametër hyrës (`in`) ky atribut merr vlerën 1 (rreshti 16), për atë dalës (`out`) merr vlerën 2 (rreshti 17), ndërsa për atë hyrës/dalës (`inout`) merr vlerën 3 (rreshti 18). Atributi `type` përmban tipin e parametrit. Atributi `size` përmban madhësinë në bajtë, ndërsa atributi `name` përmban emërtimin e parametrit.

Klasa `IDLEExceptionList` (shtojca E.12) është fare e thjeshtë. Ajo përmban gabimin aktual, atributi `exception`, dhe atributet e treguesit të gabimit paraprak `previous` dhe atij vijues `next`.

Klasa `IDLEException` është e përcaktuar si në shtojcën E.13. Kuptimi i atributeve të klasës `IDLEException` është evident. Atributi `excptID` paraqet numrin identifikues te gabimit, që e shkëmbejnë serveri lidhës dhe ai në pajisje të mbyllur (mikrokontrollor). Për gabim global, ai merr vlerat nga 1 deri në 128, ndërsa

për gabim lokal nga 129 deri në 255. Atributi `excptName` paraqet emërtimin e gabimit, që përdoret nga serveri lidhës. Ndërsa, `firstParam` dhe `params` paraqesin parametrin e parë dhe të fundit të gabimit, respektivisht.

Elemente tjerë të ndërfaqes janë edhe operacioni njëkahor dhe atributi, me klasat përkatëse `IEOnewayOperation` dhe `IEAttribute`, respektivisht. Klasa `IEOnewayOperation` është dhënë në shtojcën E.14.

Edhe kuptimi i atributeve të klasës `IEOnewayOperation` është evident. Atributi `operationID` paraqet emërtimin identifikues të operacionit njëkahor. Ndërsa, `firstParam` dhe `params` paraqesin treguesit e parametrit (argumentit) të parë dhe të fundit të operacionit, respektivisht. Siç e kemi përmendur edhe më herët, operacioni njëkahor ka vetëm parametra hyrës (`in`).

Klasa `IEAttribute` është dhënë në shtojcën E.15. Kuptimi i atributeve të klasës `IEAttribute` është si vijon. Atributi `isReadOnly` përmban informatën nëse atributi është vetëm i lexueshëm. Nëse është vetëm i lexueshëm, pra kur e ka atributi vlerën `true`, atëherë gjenerohet kodi vetëm për lexim të atributit nga kujtesa e pajisjes së mbyllur (kujtesa RAM e mikrokontrollorit). Nëse është `false`, atëherë, përpos kodit për lexim të vlerës, gjenerohet edhe kodi për vendosje të vleres së atributit. Atributi `attribType` përmban të dhënat mbi tipin dhe madhësinë e atributit, si dhe vetë vlerën e atributit përmes fushave përkatëse tanimë të përmendura të klasës `Parameter`. Atributi `attributeID` përmban emrin e atributit (të përcaktuar në skedarin IDL).

5.3.2 Modifikimi i parserit

Me ekzekutimin e veglës kompilatorike Coco/R, duke ia dhënë si hyrje gramatikën në listingun 3.1 me aksionet semantike në seksionin 5.3.1, gjenerohen skaneri dhe parseri. Çka duhet bërë me këtë rast është modifikimi i parserit. Në mesin e metodave që gjenerohen janë edhe `simpletypesplusvoid` dhe `simpletypes`, që gjenerohen nga transformimet me emër të njëjtë. E para `simpletypesplusvoid` brenda saj thirr edhe metodën e dytë `simpletypes` për kthim të numrave identifikues të tipave themelor. Kështu, e dyta kthen numrat

identifikues të të gjthë tipave themelor pa atë `void`. E para kthen numrin identifikues edhe të `void`, ndërsa të tipave themelor me ndihmën e metodës së dytë.

Metoda `simpletypesplusvoid` duhet modifikuar ashtu që të duket si në shtojcën F.1. Rreshtat e prekur janë: 2, 4, 6 dhe 9. Është deklaruar variabla `tid` dhe i është ndarë vlera varësisht nga tipi i të dhënës. Kjo vlerë për tipat themelorë merret nga metoda `simpletypes` (rreshti 4) ndërsa për `void` bëhet 1 (rreshti 6). Në fund metoda e kthen vlerën e kësaj variable, pra numrin identifikues të tipit.

Metoda simpletypes duhet modifikuar ashtu që të duket si në shtojcën F.2. Rreshtat e prekur janë: 2, 5, 10, 15, 20, 25, 30, 35, 39, 42, 51, 54, 57 dhe 65. Edhe në këtë rast deklarohet variabla tpid (rreshti 2), dhe varësisht nga tipi merr vlerën e caktuar (rreshtat 5, 10, 15, 20, 25, 30, 35, 39, 42, 51, 54, 57). Në fund kjo vlerë kthehet nga metoda `simpletypes` (rreshti 65).

5.3.3 Klasa kryesore ku thirren gjeneratorët

Me gjenerimin e skanerit dhe parserit, dhe më pas modifikimin e metodave `simpletypesplusvoid` dhe `simpletypes` të parserit sipas seksionit 5.3.2, duhet implementuar klasën kryesore. Në klasën kryesore instancohen skaneri dhe parseri i modifikuar, dhe më pas instancohen edhe klasat e gjeneratorëve ku thirren edhe metodat përkatëse për gjenerim të serverit në pajisje të mbyllur (mikrokontrollor) dhe serverëve lidhës për JacORB dhe JAX-WS. Klasa kryesore e realizuar në Java, implementon metodën `main` ku aplikacioni i gjenerimit të kodit është realizuar i tipit konsolë. Ai pranon si argumente parametrat e gjenerimit, siç janë p.sh.: emërtimi dhe vendi i skedarëve IDL dhe atij implementues, gjenerimi i detyrueshëm ose jo i numrave identifikues të ndërfaqes dhe operacionit, frekuenca e klokut të mikrokontrollorit, shpejtësia e komunikimit, etj. Nëse ka gabime në argumentet hyrëse, shtypen informatat e detajuara mbi formatin e tyre dhe dilet nga aplikacioni. Nëse nuk ka gabime, vazhdohet me instancimin e klasës së skanerit të skedarit IDL, duke ia dhënë si hyrje vetë skedarin IDL (emërtimin dhe vendin). Më pastaj instancohet parseri, duke ia dhënë si hyrje skanerin e instancuar më parë. Pas instancimit të parserit, ekzekutohet metoda `Parse` e tij, me të cilën behët parsimi i skedarit IDL. Nëse ka gabime gjatë parsimit, shtypet mesazhi përkatës dhe dilet nga

aplikacioni. Nëse nuk ka gabime, instancohet klasa e skanerit të skedarit implementues, duke ia dhënë si hyrje skedarin implementues. Më tej veprohet njëjtë sikur me skanerin dhe parserin e skdeës IDL. Nëse ka gabime, dilet nga aplikacioni. Nëse nuk ka gabime, fillimisht bëhet ekzekutimi i metodës `decideWhatToGenerate`, duke ia dhënë si hyrje listat e objekteve të skedarëve IDL dhe atij implementues, si dhe parametrat që tregojnë nëse duhet detyrimisht të gjenerohen fushat identifikuese të ndërfaqes dhe operacionit. Me metodën `decideWhatToGenerate` përcaktohet nëse duhet gjeneruar fushat identifikuese të ndërfaqes dhe operacionit, duke i analizuar strukturat e listës së objekteve të skedarit implementues dhe atij IDL, respektivisht. Sekuencë nga metoda `decideWhatToGenerate` ku bëhet llogaritja dhe vendosja nëse duhet të gjenerohet fusha e numrit identifikues të operacionit për ndërfaqen e dhënë, është paraqitur në shtojcën G.1.

Në rreshtin 1 inicohet në vlerën 0 variabla `noOfOperationsDeclared`, që përmban informatën mbi numrin e përgjithshëm të operacioneve, që do të gjenerohen brenda ndërfaqes. Kjo vlerë bëhet zero për secilën ndërfaqe, para fillimit të llogaritjeve. Pastaj, për çdo operacion standard kjo vlerë rritet për 1 (rreshti 10). E njëjta vlen edhe për operacion njëkahor (rreshti 13). Për atributet, shikohet nëse ai nuk është vetëm i lexueshëm (rreshti 17). Nëse po, ekzekutohet rreshti 18, që rrit për 2 numrin e tërësishëm të operacioneve. Kjo për arsye se, në këtë rast gjenerohen dy operacione: i leximit dhe i vendosjes së atributit. Nëse jo, ekzekutohet rreshti 20, që e rrit për një numrin e tërësishëm të operacioneve. Kjo pasi që, gjenerohet vetëm operacioni i leximit të atributit. Në fund (rreshti 26), shikohet nëse numri i përgjithshëm i operacioneve është më i madh se një, ose, nëse duhet gjeneruar fusha në fjalë pavarësisht nga numri i operacioneve. Ky vendim ruhet në atributin përkatës të ndërfaqes (`i.generateOperationID`).

Pas ekzekutimit të metodës `decideWhatToGenerate` vazhdohet me gjenerimin e kodit. Klasat gjeneratore të kodit të serverit në mikrokontrollor 8051, serverëve lidhës për JacORB dhe Shërbime Ueb në Java janë `CodeGenerator8051Server2`, `CodeGeneratorJacORBBindServer2` dhe `CodeGeneratorJavaWSBindServer2`, respektivisht. Gjenerimi bëhet duke

instancuar klasat e përmendura me parametrat përkatës dhe ekzekutimin e metodës `generate` të secilës nga to.

5.3.4 Gjeneratori i serverit 8051

Gjenerimi i kodit të serverit në mikrokontrollorin 8051 (pajisje të mbyllur) bëhet përmes klasës `CodeGenerator8051Server2` të zhvilluar posaçërisht. Struktura e klasës në fjalë është dhënë në shtojcën H.1.

Nga listingu në shtojcën H.1 mund të kuptohet se përpos konstruktorit vetëm metoda generate është e deklaruar si publike. Kjo është në të vërtetë edhe menyra e të gjeneruarit të serverit ne mikrokontrollor. Thjesht, instancohet klasa `CodeGenerator8051Server2` me vlerat e dëshiruara të parametrave, dhe thirret metoda `generate` e instancës. Njëri nga parametrat e konstruktorit, ai i listës së objekteve të skedarit implementues, duhet ndërtuar ashtu siç është shpjeguar në seksionin paraprak (5.3.3). Në konstruktor, përmes metodës `preCalculations` provohet nëse tejkalohet numri i lejuar i entiteteve (ndërfaqeve dhe operacioneve) në 256, si dhe gjatësia maksimale prej 128 bajtëve e përdorimit të RAM-it. Nëse tejkalohen këto vlera, bëhet ndërpreja e gjenerimit të kodit dhe dilet nga aplikacioni.

Në shtojcën H.2 është dhënë sekuencë nga metoda `preCalculations` ku bëhet llogaritja e gjatësisë maksimale të mesazhit kërkesë (hyrës) dhe trupit, gjegjësisht parametrave kthyes të mesazhit përgjigje (dalës). Llogaritja e vetëm gjatësisë së trupit per mesazhin përgjigje është valide, pasi që *header*-i i mesazhit dërgohet nga kujtesa programuese. Nga ana tjetër, mesazhi kërkesë i tëri vendoset në RAM me rastin e pranimit.

Në rreshtin 1 (shtojca H.2) merret referenca e operacionit standard nga lista e objekteve dhe ruhet në variablën `stop`, përmes së cilës i qasemi atributeve të operacionit standard. Në rreshtat 2 dhe 3 inicializohen variablat e gjatësive të mesazhit hyrës (`stopInMsgLen`) dhe dalës (`stopOutMsgLen`) me vlerat fillestare, respektivisht. Vlera fillestare e mesazhit hyrës (kërkesës) inicializohet në 1 + bajti opsional i ndërfaqes + bajti opsional i operacionit (rreshti 2). Në rreshtin 4, nëse operacioni kthen vlerë, gjatësia e kësaj vlere i kontribuon gjatësisë së mesazhit dalës (përgjigjes). Në rreshtat 5 deri në 13 shqyrtohen të gjithë parametrat e

operacionit. Ata hyrës dhe hyrës dalës (rreshtat 8 dhe 9) i kontribuojnë gjatësisë së mesazhit kërkesë, ndërsa ata dalës (rreshtat 10 dhe 11) i kontribuojnë gjatësisë së mesazhit përgjigje. Në rreshtat 14 dhe 15 shikohet nëse këto gjatësi janë më të mëdha se ato maksimale të kalkuluara deri më tani. Deri këtu janë llogaritur vetëm gjatësitë e operacioneve që përfundojnë normalisht, pa gabime. Në rreshtat 19 deri në 34 llogariten trupat e gjatësive e të gjithë mesazheve gabim që mund të kthejë operacioni standard, nëse ai është deklaruar që mund të sinjalizojë gabime.

Kalkulimet e mesazheve të operacionit njëkahor dhe atributit janë dukshëm më të thjeshtë. Operacioni njëkahor mund të ketë vetëm mesazh kërkesë duke pasur kështu vetëm parametra hyrës. Atributi mund të jetë vetëm i lexueshëm ose standard. Rrjedhimisht mund të jenë dy palë mesazhesh të shëkmbyera. Sekuenca e parakalkulimeve të atributit është dhënë në shtojcën H.3.

Duhet theksuar se në shtojcën H.3 në rreshtin 13 bëhet zhvendosja e *offset*-it të kujtesës së rezervuar për atribute për madhësinë e tipit të atributit në fjalë. Kjo ndikon në zvogëlimin e kujtesës së lirë për pranim/dërgim të mesazheve, si dhe të kujtesës shtesë të nevojshme nga kodi i operacioneve. Pjesa përpara rreshtit 13 bën kalkulimin e gjatësive ngjashëm me operacionin standard, pasi edhe atributet trajtohen si operacione standarde, por që nuk mund të sinjalizojnë gabime. Në rastin e atributit, thjeshtësia qëndron në atë se te mesazhi përgjigje i leximit të atributit dhe mesazhi kërkesë i vendosjes së atributit, kemi vetëm një parametër të shkëmbyer. Ky parametër është vetë vlera e atributit. Te mesazhi kërkesë i leximit dhe mesazhi përgjigje i vendosjes nuk kemi fare parametër.

Pas kalimit me sukses të parakalkulimeve (metodës `preCalculations`) vazhdohet me ekzekutimin e metodës `generate`, që paraqet metodën kryesore nga thirren metodat tjera të nevojshme për kryerje me sukses të gjenerimit të kodit të serverit në mikrokontrollorin 8051. Metoda `generate`, shikuar nga struktura e kodit, përbëhet nga pjesët thuaja statike që gjenerojnë fillimin dhe fundin, dhe pjesën dinamike që gjeneron mesin e kodit të serverit në mikrokontrollor. Listingu në shtojcën H.4 paraqet sekuencë nga metoda në fjalë.

Në shtojcën H.4 rreshtat 2 deri në 7 paraqesin sekuencë të kodit që gjeneron një pjesë të fillimit, rreshtat 9 deri në 30 sekuencën që gjeneron mesin, ndërsa rreshtat

32 deri në 45 sekuencën që gjeneron një pjesë të fundit të kodit të serverit në mikrokontrollorin 8051. Pjesa e fillimit është gati statike, përveç për shembull të rreshtit 6, ku vlerat për `this.TH1` dhe `this.baudRate` tanimë janë kalkuluar në konstruktor. Rreshti 9 paraqet fillimin e pjesës dinamike që gjeneron mesin e kodit. Aty provohet nëse duhet të gjenerohet bajti i numrit identifikues të ndërfaqes. Nëse po, rreshtat 11 deri në 13 gjenerojnë leximin e bajtit nga adresa e pritur në RAM përmes adresimit indirekt dhe vendosjen e tij në akumulatorin A. Në rreshtat 16 deri në 30 gjenerohet kodi për secilën ndërfaqe të deklaruar në skedarin implementues. Konkretisht, metoda `processInterface` merret me ndërfaqen e dhënë, të zgjedhur me rreshtat 16, 17, 19 dhe 29, ku si dalje kthen *header*-ët e mesazheve të operacioneve dhe atributeve të ndërfaqes në fjalë që analizohet. *Header*-ët e gjeneruar ruhen në variablën `DB_contents`, ku hyjnë në punë në vijim në pjesën e fundme të kodit, respektivisht në rreshtat 42 dhe 43.

Metoda `processInterface` është paraqitur në shtojcën H.5.

Në listingun në shtojcën H.5 rreshti 3 deklaron variablën `retVal` që përmban *header*-ët e mesazheve që kthehen nga ekzekutimi i metodave të gjenerimit të operacionit standard (rreshti 40) dhe atributit (rreshti 46). Rreshti 4 bën resetimin e numruesit të operacioneve (standarde, njëkahëshë dhe të atributeve) `currentOperationID` në vlerën 0. Kjo bëhet për secilën ndërfaqe të deklaruar në skedarin implementues, sepse operacionet numrohen brenda ndërfaqes të cilës i takojnë. Në rreshtin 6 shikohet nëse duhet gjeneruar kodin për provë të adresimit të ndërfaqes përmes numrit të tij identifikues (linuat 7 deri në 22). Më tej, në rreshtin 24 shikohet nësë duhet gjeneruar kodin për provë të adresimit të operacionit përmes numrit të tij identifikues. Nëse rezultati i provës është po (`true`), gjenerohet kodi për lexim të bajtit nga adresa e paracaktuar e RAM-it dhe ruajtje të vlerës së tij në akumulatorin A (rreshtat 25 deri në 31). Pas provave të adresimit të ndërfaqes dhe operacionit përmes numrave identifikues, vazhdohet me gjenerimin e kodit që është specifik për llojin e operacionit. Kështu, në rreshtin 33 adresohet elementi i parë përbërës i ndërfaqes. Derisa ai element të mos jetë null (rreshti 35) – pra derisa sa të ketë elemente, shikohet tipi i tij (rreshti 37). Nëse është i tipit 1 (rreshti 39), që do të thotë operacion standard, thirret metoda `processStandardOperation` që është

gjeneratore e kodit specifik për operacionin standard (rreshti 40). Nëse është operacion njëkahor (tipi 2, rreshti 42), thirret metoda `processOnewayOperation` për gjenerim të kodit specifik për operacion njëkahor (rreshti 43). Nëse është operacion atribut (tipi 3, rreshti 45), thirret metoda `processAttribute` për gjenerim të kodit specifik për atribut (rreshti 46). Siç do të shihet në vijim, në secilën nga tri metodat e sapo përmendura, gjenerohet kodi për provë të adresimit të operacionit përkatës, nëse atributi identifikues i gjenerimit për atë ndërfaqe (`generateOperationID`) ka vlerën true. Në rreshtin 53, rritet për një numri identifikues i ndërfaqes, që përsëritet pas përpunimit të secilit ndërfaqe. Në rreshtin 55 kthehen *header*-ët e ndërfaqes që përdoren nga listingu tanimë i analizuar në shtojcën H.4 rreshti 26.

Metodën `processStandardOperation` e përbëjnë disa pjesë, prej të cilave ka të ngjashme në mes vete. Po i paraqesim vetëm ato karakteristiket. Metoda fillon me gjenerimin e kodit të provës nëse është adresuar operacioni i dhënë standard.

Në shtojcën H.6 rreshti 1 provohet nëse duhet gjeneruar bajti i adresës së operacionit. Nëse po, në rreshtin 3 gjenerohet labela që tregon se fillohet me provën e adresimit. Në rreshtin 4 bëhet gjenerimi i vetë provës së krahasimit të vlerës së akumulatorit A (vleröen e lexuar) me vlerën e pritur (`this.currentOperationID`). Në rreshtat 8 deri në 11, gjenerohet kodi i vazhdimit të provës ose jo, për rastin kur nuk është adresuar operacioni në fjalë, në varësi nëse ka akoma operacione për t'u provuar (rreshti 11) ose jo (rreshti 9).

Më pastaj kalkulohen dhe shtypen adresat e pritura të parametrave hyrës dhe hyrës/dalës të operacionit standard si në shtojcën H.7.

Në listingun në shtojcën H.7 në rreshtin 2 kalkulohet adresa e fillimit të parametrit të parë hyrës ose hyrës/dalës (`actualInPrmRAMLoc`) e cila shtypet në rreshtat 15 ose 17, që varësisht nga madhësia e tij i përshtatet edhe teksti përcjellës. Në rreshtin 18, vlera e variablës së pozitës `actualInPrmRAMLoc` rritet për vlerën e madhësisë së parametrit të posa shtypur, që paraqet edhe adresën e fillimit të parametrit vijues hyrës ose hyrës/dalës në RAM. Në rreshtin 6 shikohet nëse është parametër hyrës ose hyrës/dalës. Pasi parametrat hyrës kanë numrin e drejtimit 1, dhe

ata hyrës/dalës 3, kur shprehen këto vlera në numra binarë shihet se të dy të përmbajnë shifrën me peshë më të vogël të barabartë me 1. Ky fakt përdoret për provë në rreshtin 6, pra, nëse vlera e drejtimit & (dhe) binar me vlerën 1 është më e madhe se 0. Kod i ngjashëm gjenerohet edhe për parametrin kthyes, dhe sërish për parametrat hyrës/dalës dhe ata dalës, që për shkaqe ngjashmërie me kodin në shtojcën H.7 nuk po i paraqesim.

Nëse operacioni mund të sinjalizojë gabime, gjenerohet komenti për mënyrën si të sinjalizohen gabimet përmes sekuencës së paraqitur në shtojcën H.8.

Pjesa e komenteve si të sinjalizohen gabimet gjenerohet vetëm nëse operacioni mund të sinjalizojë gabime, shtojca H.8 rreshti 1. Pastaj gjenerohet pjesa nëse operacioni përfundon pa gabime, rreshtat 3, 4 dhe 5. Rreshtat 7 deri në 31 merren me gjenerimin e komenteve për sinjalizim të secilit nga gabimet që mund t'i sinjalizojë operacioni. Kështu në rreshtat 11 deri në 26 gjenerohen komentet për vendet në RAM, që duhet vendosur parametrat e gabimit të marrë në shqyrtim në rershtin 9. Në rreshtat 27 deri në 29 gjenerohen komentet që duhet të dekomentohen për t'u sinjalizuar gabimi në fjalë. Në rreshtin 31 vazhdohet me shqyrtimin e gabimit të rradhës. Më pastaj gjenerohet komenti për kodin që duhet shkruar zhvilluesi i serverit. Ky kod është specifik për secilin aplikacion, pra, shkruhet kod për të cilin është i destinuar operacioni.

Në vijim gjenerohet kodi për provë nëse operacioni ka përfunduar me sukses, me kusht që operacioni është deklaruar që mund të sinjalizojë gabime. Kjo varet nga vlera e akumulatorit A (e barabartë me 0) e caktuar paraprakisht, sipas sugjerimit të komenteve të gjeneruar. Prova bëhet me qëllim që të ndërtohet dhe gjenerohet kodi për dërgim të *stream*-it përkatës përgjigje serverit lidhës. Pas kësaj, bëhet prova nëse operacioni ka përfunduar me gabim, duke e gjeneruar kodin për krahasim të vlerës së akumulatorit A me vlerat e paracaktuara të gabimeve (globale dhe lokale), që është prej 1 deri në 255. Sikur edhe në rastin e përfundimit me sukses, edhe këtu gjenerohet kodi për ndërtim të *stream*-it përmes asembler instruksionit `DB`, si dhe dërgim të *stream*-it të ndërtuar. Në shtojcën H.9 është dhënë sekuenca që e gjeneron provën nëse operacioni përfundon me gabim. Pjesa që përfundon me sukses, që gjenerohet para kësaj, është plotësisht e ngjashme dhe prandaj nuk është paraqitur.

Në rreshtin 1 (shtojca H.9) adresohet e tërë lista e gabimeve që mund të gjenerohen nga operacioni i dhënë standard. Derisa kjo listë, gjegjësisht anëtarët e saj nuk janë bosh (rreshti 2) merret fusha që përmban gabimin e dhënë (rreshti 4). Në rreshtin 7, madhësia në bajtë e të gjithë parametrave të gabimit `totalExcPrmSize` inicohet në vlerën 0. Pastaj në rreshtat 8 deri në 17 llogaritet madhësia e të gjithë parametrave të mesazhit (rreshti 14) që njëkohësisht paraqet edhe trupin e mesazhit të gabimit. Në rreshtin 19 gjenerohet labela që tregon fillimin e provës nëse është adresuar operacioni i caktuar. Kështu, në rreshtin 20 bëhet krahasimi i vlerës së akumulatorit A, që përmban statusin e përfundimit të operacionit, dhe njëkohësisht treguesin e suksesit (0) ose të gabimit (të ndryshëm prej 0). Rreshti 27 paraqet hyrje në gjenerimin e kodit të kthimit të përgjigjes për gabimin e dhënë. Rreshti 28 gjeneron komentin që tregon se kodi në vijim paraqet dërgimin e *header*-it të përgjigjes së gabimit. Në rreshtin 31 bëhet ndërtimi i *header*-it të përgjigjes, që do të shkruhet në fund të kodit të serverit nga pjesa përfundimtare thuaja statike e metodës `generate`. Në rreshtin 33 shikohet nëse gabimi ka parametra, pra, nëse mesazhi ka trup. Në rast se po (rreshtat 35 deri në 37), gjenerohet kodi për dërgim të trupit të mesazhit nga RAM-i. Në rreshtin 39, pasi të jetë kthyer përgjigjja, gjenerohet kodi për kthim në pozitën fillestare – programin kryesor gjegjësisht në atë të pritjes për kërkesa të reja.

Pjesa e gjenerimit të kodit të kthimit të përgjigjes kur nuk ka gabime është më e thjeshtë se ajo kur ka gabime. Kjo për arsye se këtu nuk provohet/kthehet statuti i përfundimit të operacionit. Këtë pjesë nuk po e paraqesim për shkak të ngjashmërisë dhe thjeshtësisë me pjesën kur mund të ndodhin gabime.

5.3.5 Gjeneratori i serverit lidhës për JacORB

Gjenerimi i kodit të serverit lidhës për implementimin në Java të CORBA-s JacORB bëhet përmes klasës `CodeGeneratorJacORBBindServer2`. Struktura e klasës në fjalë është dhënë në listingun I.1.

Edhe në këtë rast, pasi të jetë instancuar klasa `CodeGeneratorJacORBBindServer2`, gjenerimi i kodit të serverit lidhës për JacORB bëhet përmes metodës së vetme publike `generate`. Metoda `generate` për secilën ndërfaqe të skedarit implementues gjeneron nga një klasë të serverit

lidhës. Kjo klasë më pas duhet të instancohet, që të përdoret si server lidhës. Arsyeja pse gjenerohet klasë për secilën ndërfaqe është se ajo gjenerohet e gatshme me të gjithë parametrat e nevojshëm, duke e liruar zhvilluesin nga punët shtesë që mund të i dilnin nga konfigurimi apo përshtatja për secilin server lidhës të veçantë.

Në listingun I.2 është paraqitur sekuencë nga metoda `generate`. Në rreshtin 1 adresohet elementi i i parë i listës së objekteve të skedarit implementues, që ruhet në variablën `ie_call`. Nëse kjo variabël nuk është `null`, gjë që provohet në rreshtin 2, vazhdohet më tej me provën tjetër, rreshti 4, nëse ky element është i llojit 1 d.m.th. ndërfaqe. Pra, duhet të jetë ndërfaqe për t'u vazhduar më tej me ekzekutimin e rreshtave 5 deri në 60. Në rreshtin 6, nga elementi `ie_call` merret objekti konkret që ai përmban – ndërfaqja, dhe ruhet në variablën `intrfc`. Në rreshtin 7, numri identifikues i ndërfaqes kthehet në string numerik nëse ai duhet gjeneruar, ose në string bosh nëse nuk duhet gjeneruar. Kjo do të duhet në rreshtin 8 ku krijohet skedari në Java me emër përkatës dhe rreshtin 12 për krijim të klasës me emërtim si të skedarit, konform kërkesave të Java-s dhe implementimit të CORBA-s JacORB. Në rreshtin 14 gjenerohet atributi i komunikimit serial i klasës së serverit lidhës. Në rreshtin 15 provohet nëse duhet gjeneruar atributi i numrit identifikues të ndërfaqes. Ky atribut opsional më vonë do të përdorej nga operacionet e ndërfaqes për të ndërtuar mesazhet kërkesë, dhe për të provuar nëse është kthyer mesazhi nga ndërfaqja e kërkuar. Në rreshtin 17 gjenerohet atributi që përmban vendimin mbi mënyrën e lidhjes me portën seriale. Ky vendim lidhet me atributin e ndërfaqes (`generateInterfaceID`), që indikon edhe nëse më shumë se një server lidhës të gjeneruar do ta ndajnë (përdorin) të njëjtën portë seriale. Në rreshtat 20 deri në 32 gjenerohet konstruktori me parametra të specifikuar. Në rreshtat 34 deri 54 bëhet analiza e ndërfaqes në fjalë. Kështu, në rreshtin 34 referencohet elementi i saj i parë. Derisa ai element nuk është `null` (rreshti 35), shih tipin e tij (rreshti 40). Nëse ai është i tipit 1 (operacion standard) (rreshti 42) thirr metodën përkatëse `processStandardOperation` (rreshti 43). Nëse është i tipit 2 (operacion njëkahor) (rreshti 45) thirr metodën përkatëse `processOnewayOperation` (rreshti 46). Nëse është i tipit 3 (atribut) (rreshti 48) thirr metodën përkatëse `processAttribute` (rreshti 49). Në rreshtin 53 vazhdohet me elementin tjetër të

ndërfaqes, i cili përpunohet si më parë duke filluar nga rreshti 35. Në rreshtin 59 numri identifikues i ndërfaqes rritet për 1, që do të thotë se atributi `interfaceID` bëhet i gatshëm për ndërfaqen e rradhës. Përfundimisht, në rreshtin 62 kalohet te ndërfaqja e rradhës, përpunimi i së cilës fillon nga rreshti 2 dhe ndiqen hapat si në rastin që analizuam deri më tani.

Nga metodat e përmendura po japim detaje të metodës `processStandardOperation`, pasi dy metodat tjera `processOnewayOperation` dhe `processAttribute` janë rast special i të parës. Në shtojcën I.3 është paraqitur fillimi i metodës `processStandardOperation`.

Në rreshtin 1 mund të shihet se metodës `processStandardOperation` nga metoda `generate` i përcillen parametrat `operation` i tipit `IEStandardOperation`, që paraqet vet operacionin standard kodi i të cilit do të gjenerohet, si dhe parametri `intrfc` i tipit `IDLEInterface`, që paraqet ndërfaqen të cilit operacioni standard i takon. Në rreshtin 3 në variablën `stopPrm` ruhet parametri i parë i operacionit standard. Në rreshtin 4, variablën `operStart` fillimi i kodit të gjeneruar të operacionit – gjegjësisht signature e tij, ndërsa në rreshtin 5, variablën `operBodyStart` fillimi i kodit të trupit të operacionit – gjegjësisht vlera që e kthen. Në rreshtin 6 (variabla `prmArgs`) argumentet e operacionit, kurse në rreshtin 7 (variabla `prmCount`) numëruesi i parametrave. Në rreshtin 8 (variabla `osOperations`) ruhen operacionet e shkrimit të parametrave dalës (hyrës për mikrokontrollorin dhe `in` ose `inout` në skedarin IDL) në *stream*-in dalës. Në rreshtin 9 (variabla `isOperations`) ruhen operacionet e leximit të parametrave hyrës (dalës për mikrokontrollorin dhe `inout` ose `out` në skedarin IDL) nga *stream*-i hyrës. Në rreshtat 10 dhe 11 kalkulohen madhësitë e *header*-ëve të mesazhit dalës dhe hyrës, që ruhen në konstantat përkatëse `osHeaderSize` dhe `isHeaderSize`. Me këto vlera më tej inicohen variablat `osSize` (rreshti 12) dhe `isSize` (rreshti 13), që paraqesin madhësitë e mesazhit dalës dhe hyrës, respektivisht. Rreshti 14 deklaron dhe inicon variablën `hasReturnVal`, që tregon nëse operacioni kthen

vlerë (jo `void`). Deklarimet e variablave që u paraqitën hyjnë në punë në kodin që vijon.

Në rreshtin 16 shikohet nëse operacioni kthen vlerë jo `void`. Nëse po, vazhdohet me ekzekutimin e rreshtave 18 deri në 22, ku variablat e posashpjeguara marrin vlera të caktuara. Kështu, në rreshtin 18 vlera e gjatësisë së *stream*-it hyrës `isSize` rritet për vlerën e madhësisë në bajtë të parametrit që e kthen operacioni. Në rreshtin 19 ndërtohet një pjesë e *signature* të operacionit. Në rreshtin 20 gjenerohet deklarimi i parametrit që kthen operacioni, duke përcaktuar tipin përkatës në Java nga ai IDL, dhe duke e caktuar vlerën e nënkuptuar për atë parametër. Në rreshtin 21 gjenerohet leximi i parametrit që kthen operacion nga *stream*-i hyrës, që ruhet në variablën e leximit të parametrave hyrës (`isOperations`). Në rreshtin 22 variabla (`hasReturnVal`) që tregon se operacioni kthen vlerë bëhet `true` logjike. Nëse operacioni nuk kthen vlerë (rreshti 25), atëherë ndërtohet vetëm *signature* e operacionit (rreshti 27), duke e specifikuar se kthen `void`. Mos prania e efektit në variablat tjera si në bllokun paraprak (rreshtat 17 deri në 23) është evident.

Kodi në vijim bën shqyrtimin e parametrave të operacionit, që variabla stopPrm tregon, duke gjeneruar operacionet e shkrimit të parametrave dalës dhe të leximit të atyre hyrës. Sekuencë nga ky kod është paraqitur në listingun I.4.

Nëse operacioni ka parametra (rreshti 1) shqyrtohet drejtimi i tij (rreshti 3). Nëse drejtimi është i llojit 3 (rreshti 8), që do të thotë është parametër hyrës/dalës (inout), vazhdohet me ekzekutimin e rreshtave 9 deri në 24. Në rreshtat 9 dhe 10 rriten gjatësitë e *stream*-it dalës dhe hyrës respektivisht, për vlerën e madhësisë së parametrit. Në rreshtin 11 numëruesi i parametrave `prmCount` rritet për 1. Kështu, në rreshtin 13 provohet nëse ky numërues është 1. Nës është 1, gjenerohet parametri i parë i *signature* të operacionit (rreshti 15). Nëse është më i madh se 1 (rreshti 17) gjenerohet i njëjti parametër, pjesë e *signature* të operacionit, por me shenjën e simbolit presë (,) përpara. Presa duhet të gjenerohet, pasi ky e pason parametrin paraprak dhe të dy ndahen me simbolin presë, konform gramatikës së IDL. Në rreshtin 22 ndërtohet kodi për shkrim të parametrit në *stream*-in dalës. Në rreshtin 23 ndërtohet kodi për lexim të të njëjtit parametër nga *stream*-i hyrës. Rastet kur drejtimi i parametrit është i llojit 1 (rreshti 4), pra dalës (në skedarin IDL si in), dhe i llojit 2

(rreshti 6) hyrës (në skedarin IDL si out) nuk janë paraqitur. Kjo për arsye se këto paraqesin raste të veçanta të llojit 3. Kështu, në rastin 1 figurojnë vetëm operacionet dalëse, ndërsa në atë 1 vetëm hyrëse. Në rreshtin 27 vazhdohet me parametrin e rradhës.

Për të mos shkaktuar konfuzion, duhet sqaruar se parametrave hyrës (`in`) të operacioneve standarde në skedarin IDL i korrespondojnë poashtu parametrat hyrës në operacionin standard të implementuar në serverin në mikrokontrollor, ndërsa në operacionin standard në serverin lidhës i korrespondojnë parametrat dalës. Vlen edhe e kundërta, parametrave dalës (`out`) të operacioneve standarde në skedarin IDL i korrespondojnë poashtu parametrat dalës në operacionin standard të implementuar në serverin në mikrokontrollor, ndërsa në operacionin standard në serverin lidhës i korrespondojnë parametrat hyrës. Sa i përket parametrave hyrës/dalës (`inout`), në serverin në mikrokontrollor ata përpunohen së pari si hyrës e pastaj si dalës, ndërsa në serverin lidhës së pari si dalës pastaj si hyrës.

Në vijim gjenerohet kodi si rezultat i përpunimit të gabimeve që mund t'i sinjalizojë operacioni standard. Kodi i gjeneruar ruhet në variablat përkatëse, që në fund shkruhen në skedarin e serverit lidhës. Sekuenca që gjeneron kodin e tillë është paraqitur në shtojcën I.5.

Në shtojcën I.5 është paraqitur kodi që ekzekutohet për secilin gabim, nga lista e gabimeve që mund të sinjalizojë operacioni standard. Secilit gabim nga lista, një nga një, i referohemi përmes instancës së gabimit `xptn`. Në rreshtin 1 numëruesi `xpcCount` i gabimeve rritet për një. Në rreshtin 2 provohet nëse ai ka vlerën 1, pra nëse është gabimi i parë që përpunohet. Nëse po, ezkekutohet kodi në rreshtin 3 ku gjenerohet pjesa e përkufizimit të operacionit që tregon se operacioni e sinjalizon gabimin në fjalë. Pra, ky është gabimi i parë, pasi i paraprin fjala e rezervuar `throws`. Tani, nëse `xpcCount` nuk e ka vlerën 1, që do të thotë se e ka më të madhe se 1, ekzekutohet kodi në rreshtin 5 që thotë: listës së gabimeve të deritashme shtoja simbolin presë (,) dhe gabimin aktual (konform rregullave të Java-s). Mund të vërehet edhe se, nëse numri identifikues i gabimit është më i madh se 128 (`xptn.excptID > 128`), që do të thotë se është gabim lokal brenda asaj ndërfaqeje, ai në listë shtohet duke i paraprirë emërtimi i ndërfaqes ngjitur me fjalën

Package dhe simbolin pikë (.), e gjith kjo në përputhje me kërkesat e teknologjisë JacORB në bazë të së cilës gjenerohet edhe serveri lidhës në fjalë. Në rreshtin 6 gjenerohet pjesa që duhet të shkruhet brenda operacionit, nëse nga serveri në mikrokontrollor vjen mesazhi me statut të ekzekutimit të ndryshëm prej zeros. Nëse statuti i ekzekutimit është i ndryshëm prej zeros, operacioni sinjalizon gabim i cili përmes mekanizmave të JacORB i manifestohet klientit. Në rreshtin 7 deklarohet variabla xpPrmTotSize që përmban gjatësinë e të gjithë parametrave të operacionit, që hyn në punë në rreshtin 24, ku gjenerohet prova nëse gabimi i përcaktuar me statutin e tij është kthyer me të gjithë parametrat e tij. Kjo bëhet duke e krahasuar gjatësinë e mesazhit të gabimit në fjalë me vlerën e parakalkuluar të tij mu përmes kësaj variable (`xpPrmTotSize`) dhe madhësisë së *header*-it të mesazhit (`isHeaderSize`). Në rreshtat 8 deri në 22 kalkulohet variabla e posapërmendur (`xpPrmTotSize`) si dhe gjenerohet pjesa e leximit të secilit parametër nga *stream*-i hyrës që i përcillet si argument instancës së klasës së gabimit në fjalë. Kjo e fundit ruhet në variable string `throwPart`. Në rreshtin 24 ndërtohet kushti që gjenerohet për të provuar nëse është lexuar mesazhi i gabimit në fjalë, që do të ketë si pasojë sinjalizimin e tij si gabim në serverin lidhës.

Pasi të jenë bërë gjenerimet e kodit në listingjet paraprake, në fund të metodës `processStandardOperation` bëhet shkrimi i të njëjtave në skedarin e serverit lidhës, brenda klasës së serverit duke e krijuar operacionin korrespondues. Sekuencë e kodit të tillë është paraqitur në shtojcën I.6.

Në rreshtin 1 shkruhet signature e operacionit në skedarin e serverit lidhës, brenda klasës së serverit. Në rreshtin 3 provohet nëse operacioni kthen vlerë, dhe nëse po shkruhet përmbajtja e variablës `operBodyStart` tanimë e paraqitur në shtojcën I.3 dhe shpjeguar. Në rreshtin 5 provohet nëse duhet gjeneruar numri identifikues i operacionit, dhe nëse po atëherë gjenerohet ai numër (rreshti 6) duke ia ndarë vlerën e variablës `currentOperationID`. Në rreshtat 7 dhe 8 gjenerohen gjatësitë e *stream*-it dalës, dhe e pritur e atij hyrës (kur operacioni përfundon pa gabim), respektivisht. Në rreshtin 10 thirret metoda `writeConnect` për gjenerim të kodit të lidhjes. Në rreshtin 12 provohet nëse duhet gjeneruar kodi për shkrim të bajtit të numrit identifikues të ndërfaqes (rreshti 13). Ngjashëm, në rreshtin 14 provohet nëse

duhet gjeneruar kodi për shkrim të bajtit të numrit identifikues të operacionit (rreshti 15). Në rreshtin 16 provohet nëse ka parametra dalës të operacionit në serverin lidhës (të llojit `in` në skedarin IDL). Nëse po përmbajtja e tyre e ndërtuar paraprakisht (`osOperations`) shkruhet në vendin e duhur brenda operacionit (rreshti 17). Në rreshtin 20 thirret metoda `writeDisconnect` për gjenerim të kodit të shkëputjes. Në rreshtin 22 gjenerohet kodi i deklarimit të *stream*-it hyrës, që krijohet nga *stream*-i i lexuar përmes portës seriale, duke ia përcjellë të njëjtin (`_ms.readStream`) si parametër. Në rreshtin 23, nga *stream*-i hyrës lexohet fusha gjithmonë prezente – ajo e gjatësisë së tij. Në rreshtin 24 sërish provohet nëse duhet gjeneruar fusha e numrit identifikues të ndërfaqes. Nëse po, ajo lexohet nga *stream*-i dhe i ndahet variablës përkatëse `_readInterfaceID`, që përdoret më pastaj për krahasim me vlerën e saj të pritur `_interfaceID`. Në rreshtin 27 gjenerohet dhe shkruhet kodi që provon nëse operacioni ka përfunduar me sukses në serverin në mikrokontrollor, duke i krahasuar vlerat e lexuara nga *stream*-i hyrës me ato të pritura. Në rreshtat 28 deri ne 33 gjenerohet kodi për lexim të parametrave kthyes të operacionit në serverin lidhës nga operacioni përkatës në serverin në mikrokontrollor, pra fjala është për parametrat e tipit `out` dhe `inout` në skedarin IDL. Në rreshtin 34 provohet nëse operacioni, gjegjësisht variabla `ifThrowCont` tanimë e paraqitur në shtojcën I.5 rreshti 24, ka përmbajtje të sinjalizimit të gabimeve. Të rikujtojmë se kjo përmbajtje është e formës, që provon nëse fushat e operacionit të lexuara nga *stream*-i hyrës kanë vlera të caktuara, dhe nëse po, atëherë të sinjalizohet gabimi duke e instancuar klasën e gabimit me argumente përkatëse. Nëse ka përmbajtje ajo shkruhet në vijim, pas dështimit të ekzekutimit me sukses të operacionit, që u paraqit më parë. Në rreshtin 37 përfundon përcaktimi i operacionit në fjalë, ndërsa në rreshtin 38 rritet për një numëruesi (numri identifikues) i operacionit, që e bën të gatshëm për përpunim të operacionit vijues.

5.3.6 Gjeneratori i serverit lidhës për Shërbime Ueb

Gjenerimi i kodit të serverit lidhës për aletrnativën në Java të Shërbimeve Ueb (JAX-WS) bëhet përmes klasës `CodeGeneratorJavaWSBindServer2`. Kjo klasë i ka të njëjtat metoda sikur edhe klasa `CORBAServerCodeGenerator2` si

dhe e ka metodën shtesë `generateException` për gjenerim të klasëve të gabimeve. Prandaj, nuk po e paraqesim fare strukturën e klasës `CodeGeneratorJavaWSBindServer2`. Në vend të saj, po i paraqesim vetëm dallimet me klasën tanimë të analizuar detajisht `CodeGeneratorJavaWSBindServer2`.

Përveç dallimeve të vogla në skedarët e gjeneruar të serverëve lidhës, që i kemi paraqitur në seksionet e strukturave të skedarëve të gjeneruar, dallimet ekzistojnë edhe në gjenerimin e skedarëve shtesë për alternativën e Shërbimeve Ueb. Skedarët shtesë që duhet gjeneruar, gjenerohen në bazë të listës së objeketeve të skedarit IDL, në vend të atij implementues që e kemi pasur të serverët lidhës. Kështu, për çdo ndërfaqe në skedarin IDL, duhet të gjenerohet ndërfaqja përkatëse konform kërkesave të JAX-WS. Poashtu, për secilin gabim në skedarin IDL, qoftë global apo lokal (i përcaktuar brenda ndërfaqes), duhet të gjenerohen dy skedarë: ai i gabimit dhe klasa *fault* e gabimit në fjalë, përmes së cilës transferohet gabimi në rrjet.

Ndërfaqet gjenerohen në metodën `generate` nga lista e objekteve të skedarit IDL. Pjesë kodi nga gjenerimi i ndërfaqeve janë dhënë në shtojcën J.1.

Në shtojcën J.1 në rreshtat 2 dhe 3 gjenerohen *annotations* që janë specifike për Shërbimet Ueb JAX-WS. Në rreshtin 4 gjenerohet përkufizimi i ndërfaqes. Në rreshtin 6 fillon analiza e ndërfaqes IDL. Derisa ka elemente (rreshti 7), shikohet lloji i elementit (rreshti 9). Nëse lloji është me vlerë 1 (operacion standard), rreshti 11, kodi që vijon në rreshtat 12 deri në 65 gjenerojnë kod që është specifik për operacion standard. Kështu, në rreshtin 13 gjenerohet *annotation* që është specifik për metodën ueb. Në rreshtin 14 merret operacioni standard si element i ndërfaqes, ndërsa në rreshtin 15 parametri i parë i tij. Në rreshtat 20 deri në 28 gjenerohet *signature* e tij, ngjashëm me atë që kemi pasur të serveri lidhës i JacORB-it. Në rreshtat 30 deri në 50 bëhet gjenerimi i parametrave (argumenteve) të operacionit. Është paraqitur vetëm rasti i gjenerimit të parametrave dalës (`out`), në rreshtat 39 dhe 42. Mund të vërehen karakteristikat e parametrave te Shërbimet Ueb JAX-WS. Ata hyrës-dalës dhe dalës fillojnë me *annotation* `@WebParam` me disa çifte çelës-vlerë (ang. key-value) si name i barabartë me `stopPrm.name` dhe mode i barabartë me `WebParam.Mode.OUT`. Në rreshtat 51 deri në 64 behët ndërtimi i listës së

gabimeve që gjeneron operacioni, që figuron në përkufizim të tij. Në rreshtat 59 dhe 61 mund të vërehet dallimi në mes të gabimeve në JacORB dhe JAX-WS. Në JAX-WS, ne vend të klasës së gabimit `xptn.excptName`, sinjalizohet klasa *fault* e gabimit që është me emrin `xptn.excptName + "_Exception"`. Në të dyja rastet, mund të figurojë edhe emërtimi i pakos që ndërtohet si emërtimi i ndërfaqes i pasuar me fjalën "`Package`", të cilës i takon gabimi nëse ai është lokal brenda asaj ndërfaqe.

Në shtojcën J.2 janë paraqitur pjesë nga kodi të metodës `generateException` që gjenerojnë klasën e gabimit dhe atë *fault*.

Në shtojcën J.2 rreshti 4 shikohet nëse gabimet janë brenda ndërfaqes (janë lokalë). Nëse po, vazhdohet me ekzekutimin e rreshtave 6 deri në 13, ku krijohet direktoria me emrin që formohet duke e bashkruar emrin e ndërfaqes dhe fjalën `Package`. Më tej, ky shtek do të përdoret për krijim të skedarit të klasës së gabimit në rreshtin 15. Në rreshtin 16 provohet nëse gabimi i takon ndërfaqes (është lokal), nëse po atëherë gjenerohet pakoja përkatëse (rreshti 18). Në rreshtin 21 gjenerohet instruksioni për importim të klasës gjenerale të gabimeve `Exception`, të cilën në rreshtin 23 e zgjeron klasa e gabimit në fjalë që gjenerohet. Në rreshtat 25 deri në 30 bëhet gjenerimi i kodit të parametrave, që për shkaqe ngjashmërie me rastet e mëparëshme nuk po e paraqesim. Në rreshtin 32 provohet nëse ka gabimi ka parametra, dhe në rast se po, bëhet shkruarja a tyre si atribute të klasës që gjenerohet. Në rreshtin 33 shtypet *signature* e konstruktorit të gabimit dhe kllapa `{`, që e bën kodin në vijim të shkruhet brenda bllokut të atij konstruktori. Më tej, në rreshtin 34 prësëri provohet nëse ka parametra, dhe nëse po, tani brenda bllokut të konstruktorit shkruhen instruksionet që vlerat e parametrave (argumentëve) të konstruktorit ia ndajnë atributeve të klasës së gabimit. Atributet e klasës së gabimit në fjalë, i kanë po të njëjtit emra si të parametrave (argumenteve) të konstruktorit të klasës. Dallimi është se atributeve iu qasemi përmes instruksionit `this.parametri`. Në rreshtat 35 dhe 36 përfundon blloku i konstruktorit dhe klasës së gabimit, respektiktivisht. Në rreshtin 37 mbyllet skedari i klasës së gabimit.

Në rreshtat 39 deri në 61 bëhet gjenerimi i klasës *fault* për klasën e gabimit të dhënë. Kjo klasë shërben për transmetim të gabimit (klasës së tij) në rrjet përmes

mekanizmave të Shërbimeve Ueb. Edhe pse nuk është paraqitur e tëra, mund të kuptohet se kjo klasë është kuazi-statike, dhe në mostër e njëjtë për të gjithë gabimet. Dallimi qëndron vetëm në disa të dhëna, ku e identifikojnë se klasa është për gabimin në fjalë.

Sa i përket klasës së serverit lidhës të Shërbimeve Ueb, ajo është thuaja e njëjtë me atë të JacORB. Dallimet i kemi theksuar në seksionin 5.2.3 dhe ato i përkasin modifikimeve të vogla që t'i përshtaten kërkesave të Shërbimeve Ueb. Modifikimet manifestohen në ndryshime të disa stringjeve në gjeneratorin e kodit të Shërbimeve Ueb, të cilat ndryshime janë aq të vogla sa që e kemi parë të arsyeshme të mos i paraqesim fare.

5.3.7 Gjeneratorët e kodit përmes bllokskemave

Për të kuptuar më mirë funksionimin e gjeneratorëve dhe logjikën e gjenerimit të kodit, në figurën 5.4 përmes bllok skemave kemi paraqitur gjeneratorët dhe lidhjen e tyre.

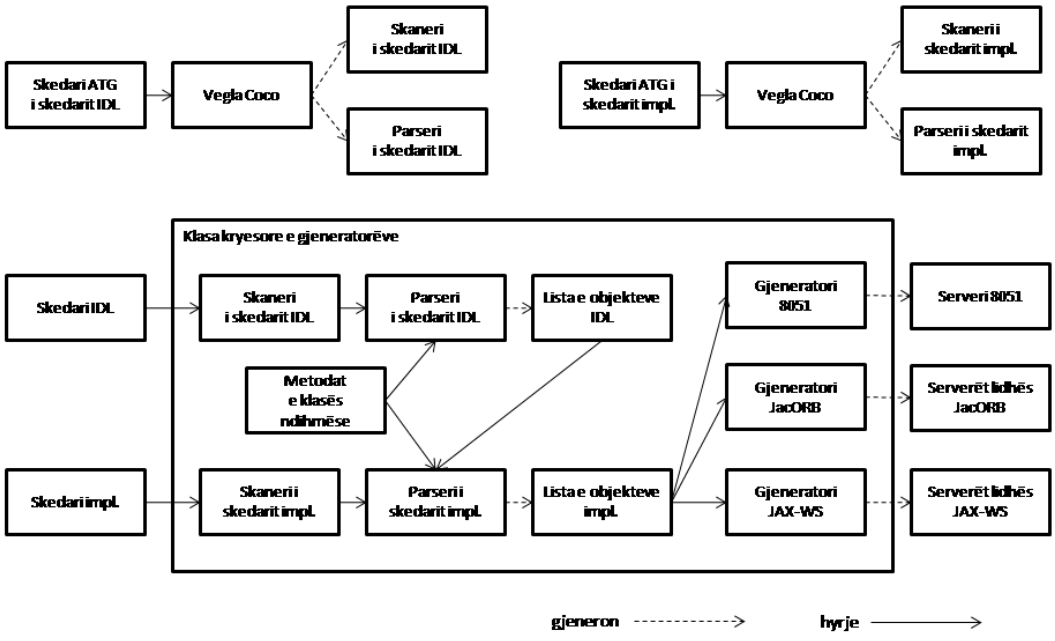


Figura 5.4 Gjeneratorët e kodit përmes bllok skemave

KAPITULLI 6

Rezultatet – madhësitë e skedarëve, mesazheve dhe kohët e komunikimit

6.1 Hyrje

Në këtë kapitull janë paraqitur rezultatet, duke marrë shembuj të ndryshëm të skedarëve IDL dhe atyre implementues si hyrje për gjeneratorët. Rezultatet maten me madhësi të skedarit që ka për t'u shkruar në mikrokontrollor dhe madhësi të mesazheve të shkëmbyera në mes të serverit lidhës dhe mikrokontrollorit të nevojshëm për ekzekutim të operacioneve. Fjala është për skedarin HEX, që fitohet pas kompilimit të atij në asembler që e gjeneron gjeneratori i kodit të serverit në mikrokontrollorin 8051. Shembujt janë marrë nga më të thjeshtit duke shkuar kah ata më kompleksë. Poashtu, janë paraqitur duke i krahasuar ngushtë me literaturën relevante edhe kohët e komunikimit klient / server në pajisje të kufizuar dhe madhësitë e mesazheve përkatëse.

6.2 Vegla për llogaritjen e madhësisë reale të kodit HEX

Në seksionin 4.2.10 kemi parë se skedari HEX ndërtohet nga rreshta të cilët përmbajnë informata për vendin dhe vetë kodin që duhet shkruar në kujtesën e kodit (kujtesën flesh programuese) të mikrokontrollorit. Figura 4.3 paraqet shembull të një rreshti në skedarin HEX. Madhësia e vërtetë e kodit që ka për t'u shkruar në mikrokontrollor është shuma e gjatësive të të gjithë rekordeve me të kod, që është e njohur edhe si gjatësi e segmentit të kodit. Kjo pasi kodi shikohet jo si tërësi fragmentesh si në skedarin HEX , por si segment i ndërtuar nga vendosja një pas një e blloqeve të tij. Enkas për këtë, ne kemi zhvilluar në aplikacion të vogël, që merr si hyrje skedarin HEX dhe shtyp si dalje gjatësinë e segmentit ose madhësinë reale të skedarit në hyrje. Aplikacioni funksionon, ashtu që lexon secilin rresht, dhe shumës i shton fushën që paraqet gjatësinë e bllokut në atë rresht. Në fund shtypet shuma totale

e gjatësive të të gjithë rreshtave, që paraqet madhësinë reale të skedarit HEX që është analizuar.

6.3 Performansat për shembuj të skedarëve IDL dhe atyre implementues

Në këtë seksion do të paraqesim madhësitë reale të skedarëve HEX dhe madhësitë e mesazheve të shkëmbyera ndërmjet serverit në mikrokontrollor dhe atij lidhës, për skedarë të ndryshëm hyrës IDL dhe implementues. Do të fillojmë nga më i thjeshti si në listingun 6.1.

```
interface Int1 {
};
```

Listingu 6.1 Skedari IDL me një ndërfaqe boshe

Ndërsa skedari implementues më minimal (jo bosh) duket si në listingun 6.2.

```
implement Int1;
```

Listingu 6.2 Skedari implementues me një ndërfaqe – ndërfaqen Int1

Me ekzekutimin e gjeneratorit të serverit në mikrokontrollor fitohet skedari në asembler, i cili pas kompilimit e ka madhësinë reale prej vetëm 89 bajtësh. Sa i përket madhësisë së mesazheve, në këtë rast nuk ka mesazhe të shkëmbyera pasi nuk ka operacione të përcaktuara.

Në interes janë edhe madhësia reale e skedarit HEX si dhe madhësitë e mesazheve kur janë të implementuar më shumë se një ndërfaqe. Rasti i implementimit të dy ndërfaqeve për ndërfaqen `Int1` është skedari implementues si në listingun 6.3.

```
implement Int1;
implement Int1;
```

Listingu 6.3 Skedari implementues me dy ndërfaqe

Për skedarin implementues si në listingun 6.3 madhësia reale e skedarit HEX është vetëm 110 bajtë. Edhe në këtë rast nuk kemi mesazhe të shkëmbyera në mes serverëve.

Rast paksa më kompleks së ai në listingun 6.1 është paraqitur në listingun 6.4.

```
interface Int2 {
   oneway void op();
};
```

Listingu 6.4 Skedari IDL me ndërfaqe me një operacion njëkahësh

Edhe në këtë rast janë me rëndësi madhësitë e skedarëve dhe mesazheve për të dyja rastet: me një dhe dy ndërfaqe të implementuar. Këto të dhëna po i paraqesim në dy tabelat 6.1 dhe 6.2, respektivisht.

Tabela 6.1 Detajet e serverit në mikrokontrollor me vetëm një ndërfaqe të implementuar, ndërfaqen Int2 nga listingu 6.4

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.4	implement Int2;	92
Ndërf. e implem.	**ID e ndërfaqes**	
Int2		
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
op	Gjatësie prej 1 bajti Bajti i parë: gjat. e mes. = 1	Nuk ka!

Nga tabela 6.1 mund të kuptohet se kemi vetëm një operacion, `op`. Gjatësia e kërkesës së ekzekutimit të operacionit op është 1 bajtëshe. Bajti i parë dhe i vetëm, është bajti i gjatësisë së mesazhit dhe e ka vlerën 1. Ky operacion është njëkahor,

prandaj nuk ka përgjigje. Madhësia reale e kodit në skedarin HEX të serverit në fjalë është vetëm 92 bajtë.

Tabela 6.2 Detajet e serverit në mikrokontrollor me dy ndërfaqe të implementuara, dy instanca të ndërfaqes Int2 nga listingu 6.4

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.4	implement Int2; implement Int2;	116
Ndërf. e implem.	**ID e ndërfaqes**	
Int2	0	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
op	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 0	Nuk ka!
Ndërf. e implem.	**ID e ndërfaqes**	
Int2	1	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
op	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 1	Nuk ka!

Në tabelën 6.2 mund të shihet kur i kemi dy ndërfaqe të implemetuara, mesazhit të operacionit `op` i shtohet edhe një fushë, e ajo është numri identifikues i ndërfaqes (ID e ndërf.). Kjo për ta dalluar nëse është adresuar operacioni `op` i ndërfaqes së parë (me ID = 0) apo të dytë (me ID = 1). Është e qartë pse nuk gjenerohet numri identifikues i operacionit, për shkak se është vetëm një operacion dhe s'ka mundësi tjetër. Madhësia reale e kodit në skedarin HEX të serverit në këtë rast është vetëm 116 bajtë.

Rast i ngjashëm me atë në listingun 6.4 është paraqitur në listingun 6.5.

```
interface Int3 {
   void op();
```

```
};
```

Listingu 6.5 Skedari IDL me ndërfaqe me një operacion standard

Madhësitë e skedarëve dhe mesazheve për të dyja rastet, me një dhe dy ndërfaqe të implementuara, po i paraqesim në tabelat 6.3 dhe 6.4, respektivisht.

Tabela 6.3 Detajet e serverit në mikrokontrollor me vetëm një ndërfaqe të implementuar, ndërfaqen Int3 nga listingu 6.5

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.5	implement Int3;	101
Ndërf. e implem.	**ID e ndërfaqes**	
Int3		
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
op	Gjatësie prej 1 bajti Bajti i parë: gjat. e mes. = 1	Gjatësie prej 1 bajti Bajti i parë: gjat. e mes. = 1

Në tabelën 6.3, për dallim nga ajo 6.1 ku është paraqitur rast i ngjashëm por me operacion njëkahor, këtu për kërkesën e operacionit të vetëm `op` e kemi edhe përgjigjen. Përgjigjja është një bajtëshe, ku bajti i vetëm me vlerë 1 paraqet gjatësinë e vet atij bajti. Madhësia reale e kodit në skedarin HEX të serverit në fjalë është vetëm 101 bajtë.

Tabela 6.4 Detajet e serverit në mikrokontrollor me dy ndërfaqe të implementuara, dy instanca të ndërfaqes Int3 nga listingu 6.5

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.5	implement Int3; implement Int3;	136
Ndërf. e implem.	**ID e ndërfaqes**	
Int3	0	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
op	Gjatësie prej 2 bajtësh	Gjatësie prej 2 bajtësh

	Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 0	Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 0
Ndërf. e implem.	**ID e ndërfaqes**	
Int3	1	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
op	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 1	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 1

Për dallim nga tabela 6.2 këtu kemi përgjigje për kërkesën e operacionit të secilës ndërfaqe të implementuar. Madhësia reale e kodit në skedarin HEX të serverit në këtë rast është vetëm 136 bajtë.

Po e shyrtojmë tani rastin e ndërfaqes me një atribut vetëm të lexueshëm të paraqitur në listingun 6.6.

```
interface Int4 {
      readonly attribute octet a;
};
```

Listingu 6.6 Skedari IDL me ndërfaqe me një atribut vetëm të lexueshëm

Madhësitë e skedarëve dhe mesazheve për të dyja rastet, me një dhe dy ndërfaqe të implementuara, po i paraqesim në tabelat 6.5 dhe 6.6, respektivisht.

Tabela 6.5 Detajet e serverit në mikrokontrollor me vetëm një ndërfaqe të implementuar, ndërfaqen Int4 nga listingu 6.6

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.6	implement Int4;	112
Ndërf. e implem.	**ID e ndërfaqes**	
Int4		
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
(lexim) a	Gjatësie prej 1 bajti Bajti i parë: gjat. e mes. = 1	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2

		Bajti i 2-të: vlera e atributit a

Në tabelën 6.5, gjatësia e përgjigjes për operacionin e leximit të atributit a është evidente. Siç mund të shihet, bajti i parë është gjatësia e mesazhit ndërsa i dyti vetë vlera e atributit a. Sa i përket kodit të serverit në mikrokontrollor, kodi i gjeneruar në këtë rast do të duhej të jetë i njëjtë me atë të operacionit standard. Por, çka vërtetë e dallon nga operacioni standard, gjë që shihet edhe në dallimin në mes të madhësive reale të skedarëve HEX, është gjenerimi automatik i kodit të leximit (zhvendosjes) të atributit nga vendi ku ruhet në RAM në vendin që duhet vendosur për dërgim poashtu në RAM. Madhësia reale e kodit në skedarin HEX të serverit në fjalë është vetëm 112 bajtë.

Tabela 6.6 Detajet e serverit në mikrokontrollor me dy ndërfaqe të implementuara, dy instanca të ndërfaqes Int4 nga listingu 6.6

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.6	implement Int4; implement Int4;	158
Ndërf. e implem.	**ID e ndërfaqes**	
Int4	0	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
(lexim) a	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 0	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: vlera e atributit a
Ndërf. e implem.	**ID e ndërfaqes**	
Int4	1	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
(lexim) a	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e ndërf. = 1	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: vlera e atributit a

Në tabelën 6.6 është e qartë shtimi i bajtit të ndërfaqes në mesazhe në krahasim me tabelën 6.5. Madhësia reale e kodit në skedarin HEX të serverit në këtë rast është vetëm 158 bajtë.

Rastin paraprak po e modifikojmë ashtu që të kemi ndërfaqe por tani me një atribut standard si në listingun 6.7.

```
interface Int5 {
      attribute octet a;
};
```

Listingu 6.7 Skedari IDL me ndërfaqe me një atribut standard

Madhësitë e skedarëve dhe mesazheve për të dyja rastet, me një dhe dy ndërfaqe të implementuara, po i paraqesim në tabelat 6.7 dhe 6.8, respektivisht.

Tabela 6.7 Detajet e serverit në mikrokontrollor me vetëm një ndërfaqe të implementuar, ndërfaqen Int5 nga listingu 6.7

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.7	implement Int5;	153
Ndërf. e implem.	**ID e ndërfaqes**	
Int5		
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
(lexim) a	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e oper. = 0	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e oper. = 0 Bajti i 3-të: vlera e atributit a
(vendosje) a	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e oper. = 1 Bajti i 3-të: vlera e atributit a	Gjatësie prej 2 bajtësh Bajti i parë: gjat. e mes. = 2 Bajti i 2-të: ID e oper. = 1

Për dallim nga tabela 6.5 këtu i kemi dy operacione, të leximit dhe atë të vendosjes së atributit, dhe rrjedhimisht edhe fushën e numrit identifikues të

operacionit (ID e oper.) në mesazhet që shkëmbehen për të dalluar cili operacion është adresuar. Madhësia reale e kodit në skedarin HEX të serverit në fjalë është vetëm 153 bajtë.

Tabela 6.8 Detajet e serverit në mikrokontrollor me dy ndërfaqe të implementuara, dy instanca të ndërfaqes Int5 nga listingu 6.7

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.7	implement Int5;	242
Ndërf. e implem.	**ID e ndërfaqes**	
Int5	0	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
(lexim) a	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 0	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: vlera e atributit a
(vendosje) a	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: vlera e atributit a	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 1
Ndërf. e implem.	**ID e ndërfaqes**	
Int5	1	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
(lexim) a	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 0	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: vlera e atributit a
(vendosje) a	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: vlera e atributit a	Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 1

Në tabelën 6.8 është e qartë shtimi i bajtit të ndërfaqes (ID e ndërf.) në mesazhe në krahasim me tabelën paraprake 6.7. Madhësia reale e kodit në skedarin HEX të serverit në këtë rast është vetëm 242 bajtë.

Po e shqyrtojmë tani rastin e një kalkulatori minimal me dy operacione themelore aritmetike që mund të sinjalizojnë gabime si në listingun 6.8.

```
exception ResultOverflow {};

interface MinCalc {
      octet sum(in octet a, in octet b) raises (ResultOverflow);
      octet prod(in octet a, in octet b) raises (ResultOverflow);
};
```

Listingu 6.8 Skedari IDL me ndërfaqe me dy operacione aritmetike që mund të sinjalizojnë gabim

Madhësitë e skedarëve dhe mesazheve për të dyja rastet, me një dhe dy ndërafqe të implementuara, po i paraqesim në tabelat 6.9 dhe 6.10, respektivisht.

Tabela 6.9 Detajet e serverit në mikrokontrollor me vetëm një ndërfaqe të implementuar, ndërfaqen MinCalc nga listingu 6.8

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.8	implement MinCalc;	216
Ndërf. e imp.	**ID e ndërfaqes**	
MinCalc		
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
sum	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e oper. = 0 Bajti i 3-të: param. a Bajti i 4-të: param. b	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e oper. = 0 Bajti i 3-të: statuti = 0 (OK) Bajti i 4-të: vlera e kthyer
		Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3

		Bajti i 2-të: ID e oper. = 0 Bajti i 3-të: statuti = 1 (ResultOverflow)
prod	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e oper. = 1 Bajti i 3-të: param. a Bajti i 4-të: param. b	Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e oper. = 1 Bajti i 3-të: statuti = 0 (OK) Bajti i 4-të: vlera e kthyer
		Gjatësie prej 3 bajtësh Bajti i parë: gjat. e mes. = 3 Bajti i 2-të: ID e oper. = 1 Bajti i 3-të: statuti = 1 (ResultOverflow)

Për dallim nga tabela 6.7 edhe këtu i kemi dy operacione, por këtu kemi mundësinë që operacioni të përfundojë edhe me gabim (ResultOverflow), përveç me status 0 (OK). Për këtë arsye në mesazhet përgjigje e kemi edhe fushën statutit (statuti) për të identifikuar si ka përfunduar operacioni. Madhësia reale e kodit në skedarin HEX të këtij serveri është vetëm 216 bajtë.

Tabela 6.10 Detajet e serverit në mikrokontrollor me dy ndërfaqe të implementuara, dy instanca të ndërfaqes MinCalc nga listingu 6.8

Skedari IDL	**Skedari implement.**	**Madhësia në bajtë**
Listingu 6.8	implement MinCalc; implement MinCalc;	372
Ndërf. e imp.	**ID e ndërfaqes**	
MinCalc	0	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
sum	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: param. a Bajti i 5-të: param. b	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: statuti = 0 (OK) Bajti i 5-të: vlera e kthyer
		Gjatësie prej 4 bajtësh

		Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: statuti = 1 (ResultOverflow)
prod	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: param. a Bajti i 5-të: param. b	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: statuti = 0 (OK) Bajti i 5-të: vlera e kthyer
		Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 0 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: statuti = 1 (ResultOverflow)
Ndërf. e imp.	**ID e ndërfaqes**	
MinCalc	1	
Operacioni	**Kërkesa**	**Përgjigj-ja(-et)**
sum	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: param. a Bajti i 5-të: param. b	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: statuti = 0 (OK) Bajti i 5-të: vlera e kthyer
		Gjatësie prej 4 bajtësh Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 0 Bajti i 4-të: statuti = 1 (ResultOverflow)
prod	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: param. a Bajti i 5-të: param. b	Gjatësie prej 5 bajtësh Bajti i parë: gjat. e mes. = 5 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: statuti = 0 (OK) Bajti i 5-të: vlera e kthyer
		Gjatësie prej 4 bajtësh

		Bajti i parë: gjat. e mes. = 4 Bajti i 2-të: ID e ndërf. = 1 Bajti i 3-të: ID e oper. = 1 Bajti i 4-të: statuti = 1 (ResultOverflow)

Në tabelën 6.10 është e qartë shtimi i bajtit të ndërfaqes (ID e ndërf.) në mesazhe në krahasim me tabelën paraprake 6.9. Madhësia reale e kodit në skedarin HEX të serverit në këtë rast është vetëm 372 bajtë.

Në vijim, për të pasur një pasqyrë edhe më të qartë, janë paraqitur varësitë e madhësisë reale të kodit pas kompilimit (skedarit HEX) dhe mesazheve nga numri i operacioneve në ndërfaqe dhe numri i ndërfaqeve. Veças janë shqyrtuar varësitë nga numri i operacioneve standarde dhe atyre njëkahore për një dhe dy ndërfaqe të implementuara.

Në figurën 6.1 është paraqitur varësia e madhësisë reale e skedarit HEX pas kompilimit nga numri i operacioneve standarde në ndërfaqe, për një dhe dy ndërfaqe të implementuara (shih legjendën në figurë për t'iu referuar grafeve përkatëse). Në boshtin horizontal (abshisë) janë paraqitur numri i operacioneve, ndërsa në atë vertikal (ordinatë) madhësia e kodit rezultues nga numri i operacioneve. Janë paraqitur dy grafe, ai i varësisë për një ndërfaqe dhe ai për dy ndërfaqe të implementuara.

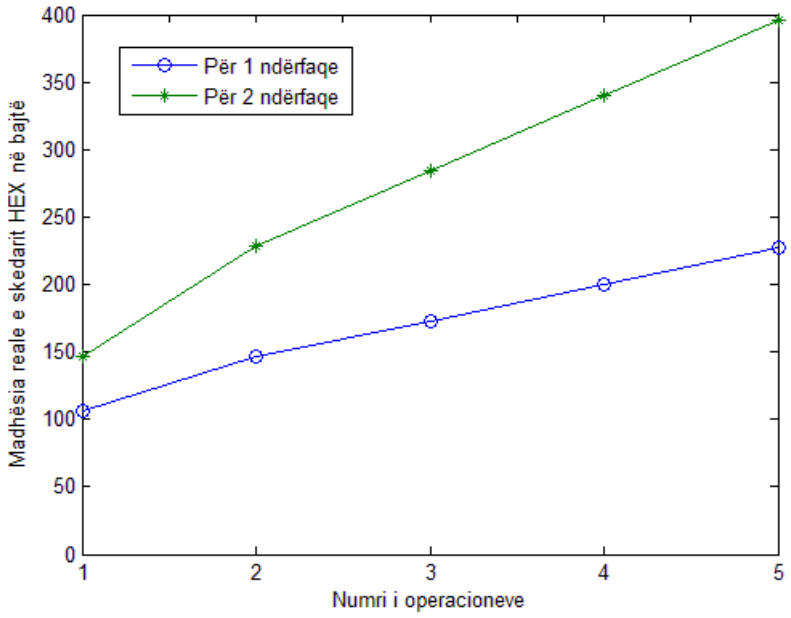


Figura 6.1 Varësia e madhësisë reale të skedarit HEX nga numri i operacioneve standarde në ndërfaqe për një dhe dy ndërfaqe të implementuara

Për operacione standarde është marrë një mostër e fiksuar e operacionit dhe është përsëritur nga një deri në pesë herë për marrje të rezultateve. Kështu, p.sh. për tri operacione standarde, skedari IDL duket si në listingun 6.9 në vijim.

```
interface TestServer {
      long stOper1(in short id);
      long stOper2(in short id);
      long stOper3(in short id);
};
```

Listingu 6.9 Skedari IDL me ndërfaqe me tri operacione standarde

Rritja më e madhe e kodit gjatë shtimit të operacionit të dytë (nga rasti me një operacion) në figurën 6.1 në krahasim me shtimin e operacionit të tretë, të katërt dhe të pestë, shpjegohet me atë se në rastin e dy operacioneve për dallim nga tre, katër dhe pesë shtohet për herë të parë edhe pjesa për lexim të numrit (bajtit) identifikues të operacionit. Kjo fushë është e re në krahasim me kur e kemi vetëm një operacion dhe shtohet kodi përkatës për lexim nga kujtesa RAM. Në secilin rast vijues, pra me dy, tre, katër dhe pesë operacione shtohet edhe pjesa për përcaktim nëse është adresuar

operacioni përkatës. Prandaj, kemi rritje proporcionale (lineare) të madhësisë, pasi shtohet kod identik. Është e qartë edhe se, grafi për dy ndërfaqe të implementuara përmban madhësi më të mëdha kodi në krahasim me një ndërfaqe të implementuar. Në këtë rast shtohet edhe pjesa e kodit për lexim të numrit (bajtit) identifikues të ndërfaqes dhe krahasim të atij numri nëse është adresuar ndërfaqja e parë ose e dytë e implementuar.

Krahas madhësive të kodit po i paraqesim edhe madhësitë e mesazheve në varësi nga numri i operacioneve standarde – në mënyrë identike si në rastin e shpjeguar mbi madhësitë e kodit. Grafi i madhësive të mesazheve është paraqitur në figurën 6.2.

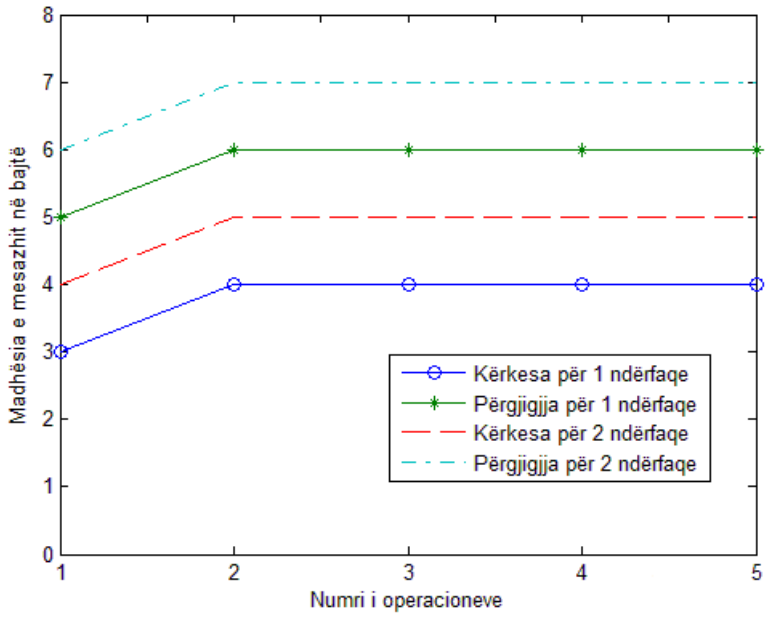
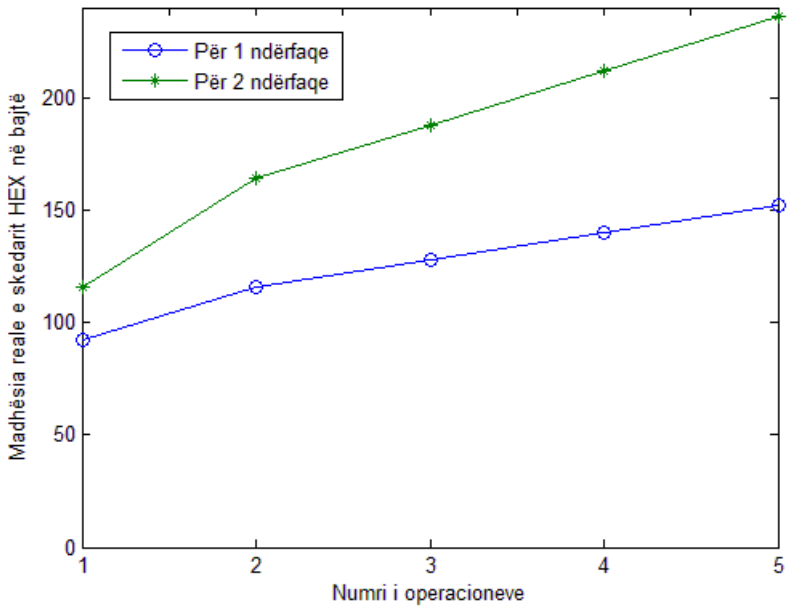


Figura 6.2 Varësia e madhësisë së mesazheve nga numri i operacioneve standarde në ndërfaqe për një dhe dy ndërfaqe të implementuara

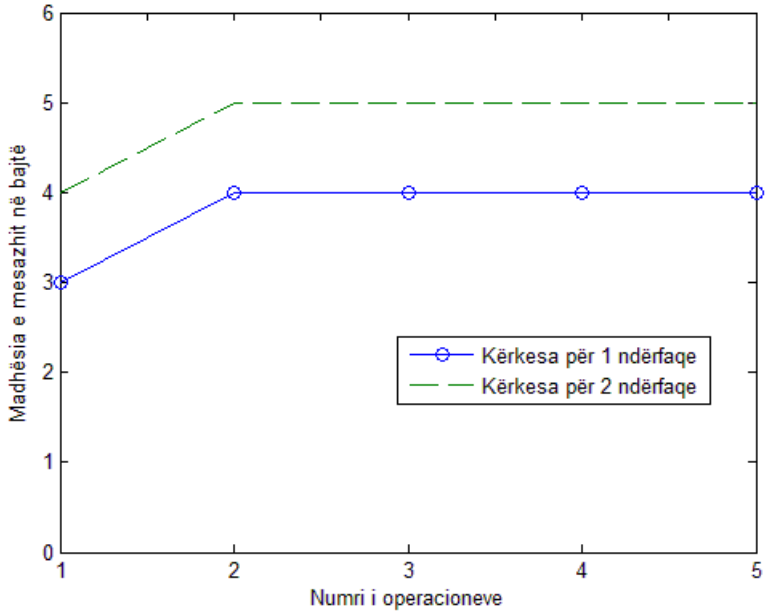
Grafet në figurën 6.2 tregojnë varësinë e madhësisë së mesazheve të shkëmbyer ndërmjet serverit lidhës dhe serverit në pajisje të mbyllur në varësi nga numri i operacioneve standardë në skedarin IDL si dhe nga numri ndërfaqeve të implementuara. Është e qartë se për dy ndërfaqe madhësia përkatëse e mesazhit kërkesë ose përgjigje është më e madhe se ajo për një ndërfaqe. Me shtimin e numrit të ndërfaqeve në më shumë se dy kjo madhësi nuk rritet, pasi mjafton bajti i shtuar për identifikimin e ndërfaqes. Duhet ritheksuar se kjo vlen deri në 256 ndërfaqe të

implementuara që paraqet njëherësh edhe vlerën maksimale që mund të ruhet në një bajt të vetëm. Shpjegim i ngjashëm vlen edhe për varësinë e madhësisë së mesazhit nga numri i operacioneve, ku mund të shihet se me rritjen e numrit të operacioneve nga një në dy rritet për një bajt madhësia e mesazhit përkatës. Kjo madhësi nuk ndryshon më tej edhe me rritjen e numrit të operacioneve në tre, katër ose pesë, pasi bajti i shtuar në fjalë mjafton për identifikim të tyre (deri në 256 sosh).

Grafe të ngjashëm fitojmë edhe për rastin e varësisë së madhësisë reale të kodit pas kompilimit dhe madhësisë së mesazheve nga numri i operacioneve njëkahore në skedarin IDL për një dhe dy ndërfaqe të implementuara, si në figurën 6.3.a dhe b.



a) Madhësia reale e kodit



b) Madhësia e mesazheve

Figura 6.3 Varësia e a) madhësisë reale të skedarit HEX, dhe b) madhësisë së mesazheve nga numri i operacioneve njëkahore në ndërfaqe për një dhe dy ndërfaqe të implementuara

Për grafet e operacioneve njëkahore, arsyetimi është i ngjashëm si në rastin e operacioneve standarde. Vetëm se, në këtë rast fitojmë madhësi më të vogla kodi, pasi te operacionet njëkahore mungon pjesa e kodit të kthimit të përgjigjes nga operacioni – të gjithë operacionet njëkahore kthejnë `void`. Edhe për madhësinë e mesazheve mund të konkludojmë në mënyrë të ngjashme.

Vlen të paraqitet edhe një rast i skedarit IDL me operacione njëkahore për të mbështetur rezultatet, si në listingun 6.10.

```
interface TestServer {
   oneway void owOper1(in short id);
   oneway void owOper2(in short id);
   oneway void owOper3(in short id);
   oneway void owOper4(in short id);
};
```

Listingu 6.10 Skedari IDL me ndërfaqe me katër operacione njëkahore

Në të dyja rastet – operacionet standarde dhe njëkahore, për konkludimin e madhësive të mesazheve, mund të na shërbejnë tabelat me detaje të mesazheve dhe fushave përkatëse për skedarë të ndryshëm IDL dhe implementues si hyrje, të paraqitura në këtë kapitull.

6.4 Performansat në krahasim me të arriturat në literaturën relevante

Tani, pasi i kemi paraqitur të detajuara në formë tabelare madhësitë dhe përmbajtjet e mesazheve të shkëmbyera ndërjmet serverit lidhës dhe atij në mikrokontrollor për skedarë të ndryshëm IDL dhe implementues hyrës, po i krahasojmë madhësitë e mesazheve me ato të prezantuara në literaturën relevante [41]. Krahas madhësive të mesazheve, po i krahasojmë edhe kohët e komunikimit klient – server në mikrokontrollor, si dhe madhësitë e skedarëve të serverit në mikrokontrollor pas kompilimit me rezultatet përkatëse në literaturën relevante.

Skema e sistemit nga literatura që krahasohet është paraqitur në figurën 6.4.

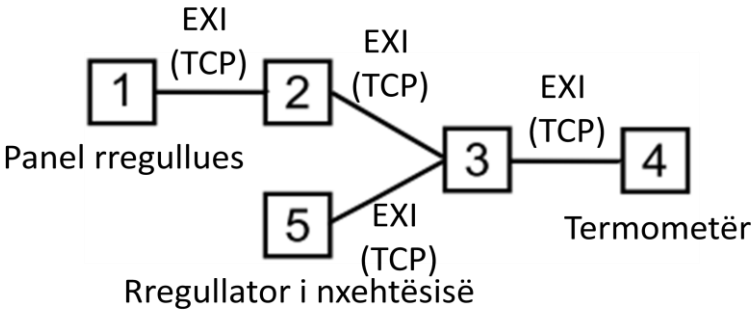


Figura 6.4 Skema e sistemit nga literatura që krahasohet

Elementet përbërëse të sistemit në figurën 6.4 janë: nyja 1 paraqet kompjuter personal (PC); nyjet 2, 3, 4 dhe 5 paraqesin mikrokontrollorë. Në këtë rast, dallojmë kohët e shkëmbimit të mesazheve ndërmjet nyjeve 1 dhe 5 (1, 5), 1 dhe 4 (1, 4) dhe 5 dhe 4 (5, 4). Mesazhet janë identike për operacionin e dhënë pavarësisht nga nyjet që i shkëmbejnë ato.

Skema e sistemit nga tonë që krahasohet me atë në literaturë është paraqitur në figurën 6.5.

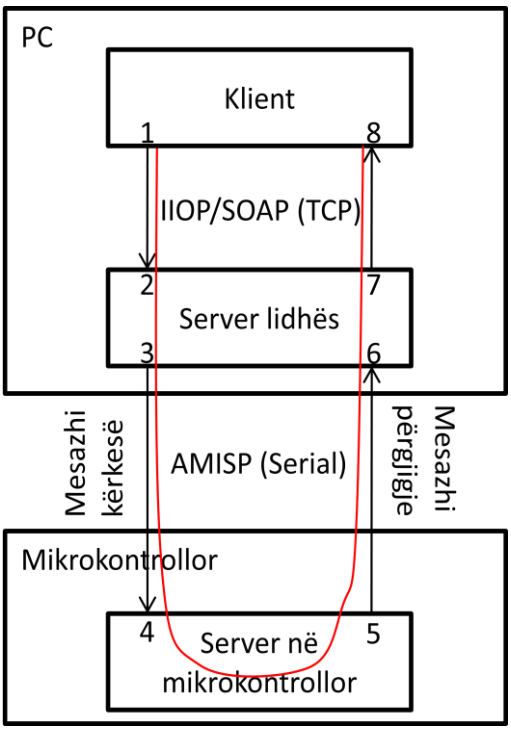


Figura 6.5 Skema e sistemit tonë që krahasohet

Skema e sistemit tonë në figurën 6.5 është skemë e rëndomtë e konfiguracionit kompjuter (PC)-mikrokontrollor. Në këtë rast, klienti dhe serveri ndodhen në të njëjtin kompjuter (blloku PC) dhe komunikojnë me serverin në mikrokontrollor (blloku Mikrokontrollor) përmes protokollit AMISP. Është me rëndësi të theksohet se cilat janë mesazhet dhe kohët e komunikimit që na interesojnë – janë matur. Kështu, mesazhi kërkesë që na intereson është ai ndërmjet pikave 3 dhe 4. Mesazhi përgjigje që na intereson është ai ndërmjet pikave 5 dhe 6. Ndërsa, koha e shkëmbimit të mesazheve është koha totale që i duhet kërkesës dhe përgjigjes, përmes pikave 1 2 3 4 5 6 7 dhe 8.

Edhe pse në literaturën krahasuese janë dhënë rezultatet për server të përcaktuar me ndërfaqe me një strukturë të caktuar, duke e marrë parasysh natyrën adaptive të kodit tonë të gjeneruar, ne po i krahasojmë rezultatet për përkufizime të ndryshme të serverëve të përcaktuar me skedarët IDL dhe implementues. Kështu për skedarin IDL mund t'i dallojmë dy raste: atë kur në ndërfaqe figurojnë të gjithë operacionet (shih listingun 6.11), të shënuar në tabelë me kolonën 'Të gjithë', dhe atë kur në ndërfaqe figuron vetëm një operacion (shih listingun 6.12), të shënuar në tabelë me kolonën 'Vetëm'.

```
interface Int8051 {
      void setTemperature(in float temp);
      float getTemperature(in octet prm);
      float getVoltage();
      void setState(in octet prm);
      octet getState();
};
```

Listingu 6.11 Skedari IDL me ndërfaqe me të gjithë operacionet

```
interface Int8051 {
      float getTemperature(in octet prm);
};
```

Listingu 6.12 Skedari IDL me ndërfaqe me një operacion

Edhe për skedarin implementues dallojmë dy raste: kur skedari implementues implementon një ndërfaqe (shih listingun 6.13), i shënuar në kolonë me ‘1’, dhe kur skedari implementues implementon dy ndërfaqe (shih listingun 6.14), i shënuar në kolonë me ‘2’.

```
implement Int8051;
```

Listingu 6.13 Skedari implementues me një ndërfaqe – ‘1’

```
implement Int8051;
implement Int8051;
```

Listingu 6.14 Skedari implementues me dy ndërfaqe – ‘2’

Matjet janë bërë për të gjitha kombinimet e mundshme të skedarëve IDL dhe atyre implementues. Prandaj, për krahasim figurojnë katër kolona të identifikuara si: 'Vetëm 1', 'Të gjithë 1', 'Vetëm 2' dhe 'Të gjithë 2'. Kuptimi i p.sh. 'Vetëm 1' është se kolona në fjalë paraqet performansën për skedarin IDL me një operacion – të

përcaktuar me rreshtin përkatës të tabelës, dhe një ndërfaqe të implementuar në skedarin implementues.

Në tabelën 6.11 në vijim po i paraqesim madhësitë e mesazheve nga literatura relevante dhe ato të fituara me gjenerimin e skedarëve me gjeneratorët e kodit që ne kemi zhvilluar. Në literaturën relevante, vlerat paraqesin madhësitë e mesazheve të shkëmbyera ndërmjet klientit dhe serverit në pajisje të mbyllur. Në rastin tonë, ato paraqesin madhësitë e mesazheve të shkëmbyer ndërmjet serverit lidhës dhe serverit në pajise të mbyllur (mikrokontrollor). Duhet theksuar se madhësitë e mesazheve në rastin e serverëve tanë janë të njëjta, qofshin ato ndërmjet serverëve lidhës për JacORB dhe mikrokontrollor, qofshin ndërmjet serverëve lidhës për JAX-WS dhe mikrokontrollor. Në të vërtetë, në të dyja rastet mesazhet e shkëmbyera janë plotësisht identike.

Tabela 6.11 Madhësitë e mesazheve të shkëmbyera klient/server në pajisje të mbyllur dhe server lidhës/server në pajisje të mbyllur

Operacioni	Mesazhi kërkesë						Mesazhi përgjigje					
	XML	EXI	Vetëm 1	Të gjithë 1	Vetëm 2	Të gjithë 2	XML	EXI	Vetëm 1	Të gjithë 1	Vetëm 2	Të gjithë 2
void setTemperature(float)	350	7	5	6	6	7	368	4	1	2	2	3
float getTemperature(byte)	333	4	2	3	3	4	351	7	5	6	6	7
float getVoltage()	231	4	1	2	2	3	335	7	5	6	6	7
void setState(byte)	306	4	2	3	3	4	321	4	1	2	2	3
byte getState()	264	4	1	2	2	3	328	4	2	3	3	4

Në tabelën 6.11 i kemi paraqitur madhësitë e mesazheve kërkesë dhe përgjigje për teknologji të ndryshme dhe deklarime të ndërfaqes për qasjen tonë përmes gjeneratorit të kodit. Kështu, kolona e hijezuar më errët ‘XML’ paraqet madhësitë e mesazheve për

operacionin konkret të përcaktuar në rreshtat e kolonës ‘Operacioni’ i implementuar përmes veglave gSOAP [14][41] të bazuar në XML. Të rikujtojmë se me këtë rast mesazhet shkëmbehen si pako XML, që është shumë e papërshtatshme për ç’gjë dëshmojnë edhe vlerat (madhësitë) e mesazheve. Kolona ‘EXI’ paraqet madhësitë e mesazheve për teknologjinë EXI të dhënë detajisht në literaturën [41]. Literatura [41] paraqet burimin kryesor për krahasim me të arriturat tona. Edhe pse struktura e serverit të paraqitur me kolonën ‘EXI’ i korespondon më shumë serverit tonë të përshkruar me ndërfaqen si në kolonën ‘Të gjithë 1’, ne i kemi gjeneruar edhe alternativat tjera për krahasim. Struktura më e ngjashme, pra ajo e paraqitur me kolonën ‘Të gjithë 1’, tregon performansa më të mira (madhësi më të volga të mesazheve) se ajo përkatëse në literaturën relevante. Edhe në rastin më të keq, atë ‘Të gjithë 2’, qasja jonë jep rezultate të njëjta ose më të mira se qasja në literaturën përkatëse të paraqitura në kolonën ‘EXI’.

Sa i përket kohëve të komunikimit, respektivisht kohëve të shkëmbimit të mesazheve kërkesë dhe përgjigje në mes të klientit dhe serverit në pajisje të mbyllur, rezultatet janë edhe më të mira. Kohët e komunikimit me serverin në mikrokontrollor nga klienti përmes serverëve lidhës të gjeneruar janë disa dhjetëra herë më të vogla se ato të raportuara në literaturën relevante [41]. Rezultatet në fjalë janë paraqitur në tabelat 6.12 dhe 6.13 në vijim.

Tabela 6.12 Kohët e komunikimit klient/server në pajisje të mbyllur për JacORB

JacORB klient - mikrokontrollor (në ms)					
Operacioni	EXI$^{(nodeID,\ nodeID)}$	Vetëm 1	Të gjithë 1	Vetëm 2	Të gjithë 2
void setTemperature(float)	$1319^{(1,\ 5)}$	53	73	72	91
float getTemperature(byte)	$1093^{(5,\ 4)}$ / $1345^{(1,\ 4)}$	61	79	81	97
float getVoltage()	$1471^{(1,\ 4)}$ / $1479^{(1,\ 5)}$	55	78	73	88
void setState(byte)	$1212^{(1,\ 4)}$ / $1199^{(1,\ 5)}$	27	50	50	65
byte getState()	$1181^{(1,\ 4)}$ / $1146^{(1,\ 5)}$	30	54	51	63

Në tabelën 6.12 janë paraqitur për krahasim kohët e komunikimit klient/server, për teknologjinë EXI në kolonën ‘EXI’nga literatura relevante [41], dhe teknologjinë

JacORB për kodin e gjeneruar nga veglat tona për kombinimet tanimë të shpjeguara të paraqitura përmes kolonave përkatëse ‘Vetëm 1’, ‘Të gjithë 1’, ‘Vetëm 2’ dhe ‘Të gjithë 2’.

Tabela 6.13 Kohët e komunikimit klient/server në pajisje të mbyllur për JAX-WS

JAX-WS klient - mikrokontrollor (në ms)					
Operacioni	EXI$^{(nodeID,\ nodeID)}$	Vetëm 1	Të gjithë 1	Vetëm 2	Të gjithë 2
void setTemperature(float)	$1319^{(1,\ 5)}$	57	77	77	94
float getTemperature(byte)	$1093^{(5,\ 4)}$ / $1345^{(1,\ 4)}$	70	81	85	101
float getVoltage()	$1471^{(1,\ 4)}$ / $1479^{(1,\ 5)}$	58	79	78	93
void setState(byte)	$1212^{(1,\ 4)}$ / $1199^{(1,\ 5)}$	32	52	50	74
byte getState()	$1181^{(1,\ 4)}$ / $1146^{(1,\ 5)}$	31	50	51	67

Ngjashëm, në tabelën 6.13 janë ri-paraqitur poashtu për krahasim kohët e komunikimit klient/server për teknologjinë EXI (kolona ‘EXI’) nga [41], me ato të gjeneruara nga veglat tona për alternativën e Shërbimeve Ueb (JAX-WS) për kombinimet përkatëse në kolonat ‘Vetëm 1’, ‘Të gjithë 1’, ‘Vetëm 2’ dhe ‘Të gjithë 2’.

Kohët janë matur në milisekonda. Lehtë mund të konkludohet se performansat tona janë dukshëm më të mira - të rendit disa dhjetëra herë më të vogla. Mund të vërehet poashtu se, performansat tona për alternativën e JacORB janë pakëz më të mira (për disa milisekonda) se ato për JAX-WS. Kjo për arsye se JacORB përdor protokoll binar ndërsa JAX-WS protokoll tekstual për komunikim klient/server lidhës.

Vlen të sqarohet se matja e kohëve të komunikimit është realizuar ashtu që, për secilën matje (qelizë në tabelë) janë bërë 25 kërkesa/përgjigje dhe është llogaritur mesatarja e kohëve të 20 kërkesave/përgjigjeve të fundit, duke i neglizhuar 5 të parat (shih figurën 6.6).

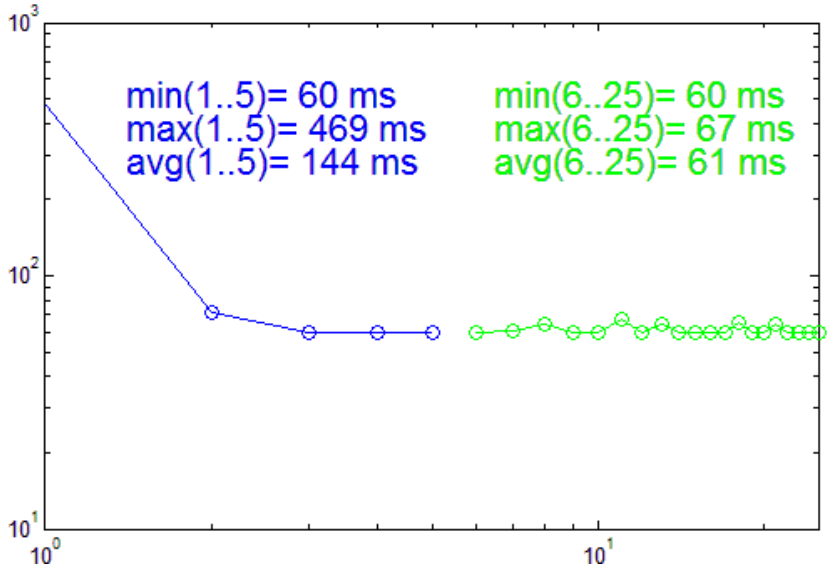


Figura 6.6 Mostrat për matje të kohëve të komunikimit klient/server në mikrokontrollor (25 mostrat e para)

Pesë (5) mostrat e para (në figurën 6.6 paraqitur me të kaltër) janë neglizhuar, sepse kur sistemi realisht punon në kohë më të gjatë ato nuk ndikojnë në mesataren e komunikimit pasi vetëm mostra e parë ka vlerë më të madhe (ka kohë më të gjatë) dhe kjo nuk ndikon në mesataren e tërësishme të kohës në rastin kur sistemi do të punonte një kohë të gjatë – që praktikisht do të ishte i destinuar. Kjo vërtetohet me grafet e 200 mostrave (të paraqitur në figurën 6.7 në vijim) të marra për operacionin e njëjtë me atë të figurës paraprake 6.6.

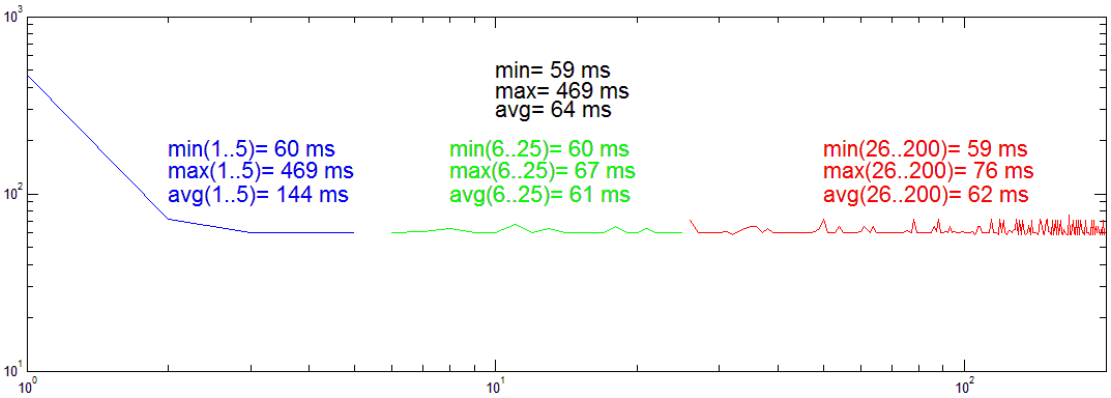


Figura 6.7 Mostrat për matje të kohëve të komunikimit klient/server në mikrokontrollor (200 mostrat e para)

Nga figura 6.7 mund të konkludohet se mesatarja e kohëve të komunikimit të 200 mostrave (në figurën 6.7 vlera avg = 64 ms me të zeza) është e përafërt me atë të

20 mostrave (në figurën 6.7 vlera avg(6..25) = 61 ms me të gjelbër) që kemi marrë ne. Ne i kemi marrë 20 mostra për të përfituar në kohë (shpejtësi) të matjeve, në krahasim me të themi për 200 mostra për të cilin rast do të fitonim rezultate të ngjashme e që do të na merrnin më shumë kohë për matje.

6.5 Përfundime mbi rezultatet

Nga tabelat e paraqitura në nënkapitullin 6.3 mund të konkludohet mbi madhësi shumë të vogla të kodit real të skedarëve HEX prej vetëm disa qindra bajtësh. Poashtu edhe gjatësia e mesazheve është mjaft e kënaqshme, prej vetëm disa bajtëve, duke shkëmbyer me atë rast vetëm informata të nevojshme të identifikimit të operacioneve dhe komunikimit të vlerave (parametrave). Rezultatet e paraqitura në tabelat në fjalë janë më të mira se ato në literaturën relevante sa i përket madhësive reale të skedarëve dhe të krahasueshme sa i përket madhësisë se mesazheve të shkëmbyera. Për shembull, [42] raporton madhësi kodi prej 40 deri në 70 kilobajtë (kB) për Shërbime Ueb të bazuar në EXI, përderisa për Shërbime Ueb të bazuara në gSOAP [14] raporton madhësi kodi prej 300 kilobajtësh. Për mesazhe të shkëmbyera [40] raporton madhësi prej 2 dhe 5 bajtësh, për servise të thjeshta të ngjashme me ato që i kemi paraqitur edhe ne.

Edhe të dhënat e paraqitura ne nënkapitullin 6.4 nga krahasimi i ngushtë me literaturën relevante [41] tregojnë performansa shumë më të mira të komunikimit (disa dhjetëra herë më të shpejta) dhe mesazhe më të vogla – në rastin më të keq të krahasueshme me ato relevante.

KAPITULLI 7

Shembull – ndërtimi i një sistemi për marrje të dhënash

7.1 Hyrje

Në këtë kapitull është paraqitur një shembull konkret i ndërtimit të sistemit për marrje të dhënash për monitorim të jetëgjatësisë së baterive, duke përfshirë të gjitha fazat që nga shkruarja e skedarit IDL e deri te ekzekutimi i sistemit.

7.2 Skedari IDL, implementues dhe kodi i operacionit të leximit

Hapi i parë që duhet bërë është shkruarja e skedarit IDL. Pastaj bëhet gjenerimi i serverit lidhës dhe atij në mikrokontrollor. Pasi të jetë gjeneruar serveri në mikrokontrollor bëhet implementimi i operacioneve të tij. Pra, shkruhet kodi i operacionit që kryen funksionin për të cilën është i projektuar operacioni. Për skedarin IDL dhe implementimin e operacioneve, duhet të dihet roli i mikrokontrollorit në këtë rast dhe infrastruktura e lidhjeve me jashtë. Në kartën elektronike të mikrokontrollorit [19], krahas komponenteve tjera gjendet edhe konvertori A/D (analog në digjital) [46], dalja dhe pinët kontrollues të të cilit janë të lidhur për pinë të caktuar të mikrokontrollorit. Në hyrje të konvertorit A/D lidhen polet e baterisë, e lidhur për shpenzues, jetëgjatësia e së cilës monitorohet. Skema e një lidhjeje të tillë është paraqitur në figurën 7.1 në vijim.

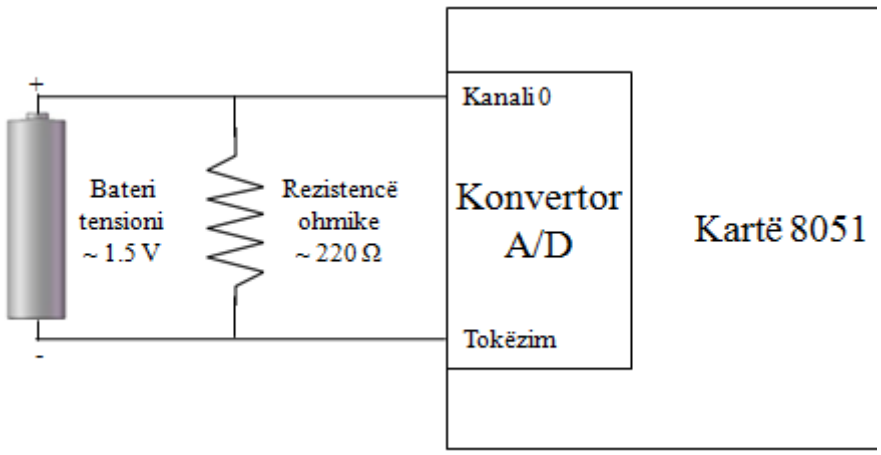


Figura 7.1 Skema e lidhjes së baterisë për konvertorin A/D të kartës 8051

Për ta përcaktuar ndërfaqen (skedarin IDL), është e nevojshme të dihen detajet e konvertorit A/D me të cilin mikrokontrollori është i lidhur dhe i lexon vlerat binare digjitale të tensionit në hyrje të tij. Konvertori A/D që është në kartë [46] mund të punojnë në modën *single-ended* dhe diferenical, dhe i ka katër kanale, që do të thotë se në të njëjtën kohë mund të lidhen katër tensione analoge në hyrje. Meqë matet tensioni i vetëm një baterie (vlera >= 0), është evidente se do të punojë në modën *single-ended*. Është lënë mundësia që të zgjidhet edhe kanali prej të cilit klienti dëshiron të lexojë. I shprehur në skedarin IDL dhe implementues, serveri në mikrokontrollor do të mund të paraqitej si në listingun 7.1.

```
exception ChannelOutOfRange {octet minChannelID; octet
maxChannelID;};
exception ChannelNotInUse {};

interface ADReader {
      short readAD(in octet channelID) raises (ChannelOutOfRange,
ChannelNotInUse);
};
```

a) Skedari IDL

```
implement ADReader;
```

b) Skedari implementues

Listingu 7.1 Përkufizimi i serverit ADReader

Serveri përbëhet nga një instancë e ndërfaqes (listingu 7.1 b), dhe IDL nga një ndërfaqe operacioni i vetëm i së cilës mund të përfundojë me sukses duke kthyer vlerën binare të lexuar nga konvertori A/D, ose mund të përfundojë me njërin nga dy gabimet e deklaruara. Pasi që vetëm kanali 0 është në përdorim, nëse klienti adreson njërin nga kanalet tjerë (1, 2 ose 3) serveri në mikrokontrollor do të sinjalizojë gabimin `ChannelNotInUse`. Nëse adreson kanalin me numër identifikues (ID) të ndryshëm nga {0, 1, 2, 3}, operacioni do ta sinjalizojë gabimin `ChannelOutOfRange` që i kthen dy parametra `minChannelID` dhe `maxChannelID`, që përmbajnë informatën mbi rangun e ID-ve të kanaleve të vlefshme – në këtë rast 0 dhe 3, respektivisht. Para se të paraqitet implementimi i operacionit të vetëm, sipas logjikës që sapo paraqitëm, duhet theksuar se madhësia reale e kodit të skedarit HEX pa implementim të operacionit është vetëm 167, ndërsa madhësia e mesazheve të shkëmbyera sillet nga 2 deri në 4 bajtë.

Në listingun K.1 është paraqitur një pjesë e implementimit të operacionit readAD. Kodi është shkruar në bazë të komenteve të gjeneruar në rreshtat 2 deri në 21 si dhe komunikimi me konvertor A/D brenda kanalit 0 është bërë në bazë të specifikacionit të dokumentit të konvertorit [46]. Kështu në bazë të komenteve, parametri hyrës `channelID` i operacionit gjendet në vendin 129 në RAM. Në rreshtat 23 dhe 24 nga RAM-i vendoset në akumulatorin A. Më pastaj në rreshtat 25, 40, 44 dhe 48 shikohet nëse është adresuar kanali 0, 1, 2 dhe 3 respektivisht. Nëse është adresuar kanali 0 ekzekutohen rreshtat 29 deri në 38. Duhet theksuar se për arsye reduktimi, rreshtat 29 deri në 38 nuk paraqesin tërë kodin për lexim nga konvertori A/D. Ai është shkurtuar për të paraqitur vetëm pjesët themelore dhe vendim-marrëse. Kështu në rreshtat 31 deri 35 (e reduktuar) bëhet leximi nga shndërruesi dhe vendosja e bajtëve te lexuar në vendet 128 deri në 129 në RAM (nuk është paraqitur kodi), ashtu siç sugjeron komenti (rreshti 6). Në rreshtin 37 vendoset vlera 0 në akumulatorin A dhe bëhet kërcimi te kthimi i përgjigjes me rreshtin 38, të përfundimit të operacionit me sukses, ashtu siç është sugjeruar në komentet e gjeneruara (rreshtat 9 dhe 10). Rreshtat 39 deri në 42, 43 deri në 46, dhe 47 deri në 50 janë plotësisht në ngjashme dhe kryejnë funksion të njëjtë, shikojnë nëse është

adresuar kanali 1, 2 ose 3. Me këtë rast, siç është sugjeruar në koment, rreshtat 18 deri në 21, vendoset vlera 2 në akumulatorin A dhe kërcehet në pjesën e kthimit të përgjigjes, të sinjalizimit të gabimit `ChannelNotInUse`. Në rreshtat 51 deri në 57, që ekzekutohen nëse nuk është adresuar asnjëri nga kanalet 0, 1, 2 ose 3, bëhet vendosja e vlerave 0 në RAM vendin 128 dhe 1 në RAM vendin 129 (përmes adresimit indirekt), ashtu siç është propozuar në koment (rreshtat 13 dhe 14, respektivisht) dhe në rreshtin 56 vendoset vlera 1 në akumulatorin A dhe në rreshtin 57 kërcehet në pjesën e kthimit të përgjigjes, ashtu siç është propozuar në koment (rreshtat 16 dhe 17). Në këtë menyrë bëhet sinjalizimi i gabimit `ChannelOutOfRange` duke e kthyer rangun e kanaleve valid prej 0 (vendi 128 në RAM) deri në 3 (vendi 129 në RAM). Pjesa e kthimit të përgjigjes, pas kërcimit nga secili operacion bëhet nga kodi i gjeneruar, për ç'gjë tanimë është shkruar detajisht.

7.3 Arkitektura e sistemit

Serveri në mikrokontrollor i zhvilluar në seksionin paraprak duhet të lexohet periodikisht dhe leximet të ruhen në bazë të dhënash. Për këtë, ne kemi zhvilluar platformë të veçantë të bazuar në JacORB (CORBA) për ta ruajtur homogjenitetin me serverin në mikrokontrollor. Të rikujtojmë se serveri në mikrokontrollor mund të komunikohet edhe si Shërbim Ueb, përmes serverit lidhës përkatës që gjenerohet bashkë me atë për JacORB. Platforma e jonë konsiston prej disa moduleve, të cilët mund të jenë edhe të shpërndarë në më shumë kompjuterë dhe vende të ndryshme (rrjet LAN ose WAN) sa i përket vendndodhjeve të kompjuterëve. Në rastin tonë, për arsye demonstrimi, qëndrojnë në një kompjuter të vetëm. Modulet janë të realizuar si serverë dhe klientë CORBA të implementuar në JacORB. Ata janë: DAQLoop, DBStorRetrSamplesDetails, DBStoringSamples dhe ADReader (shih figurën 7.2).

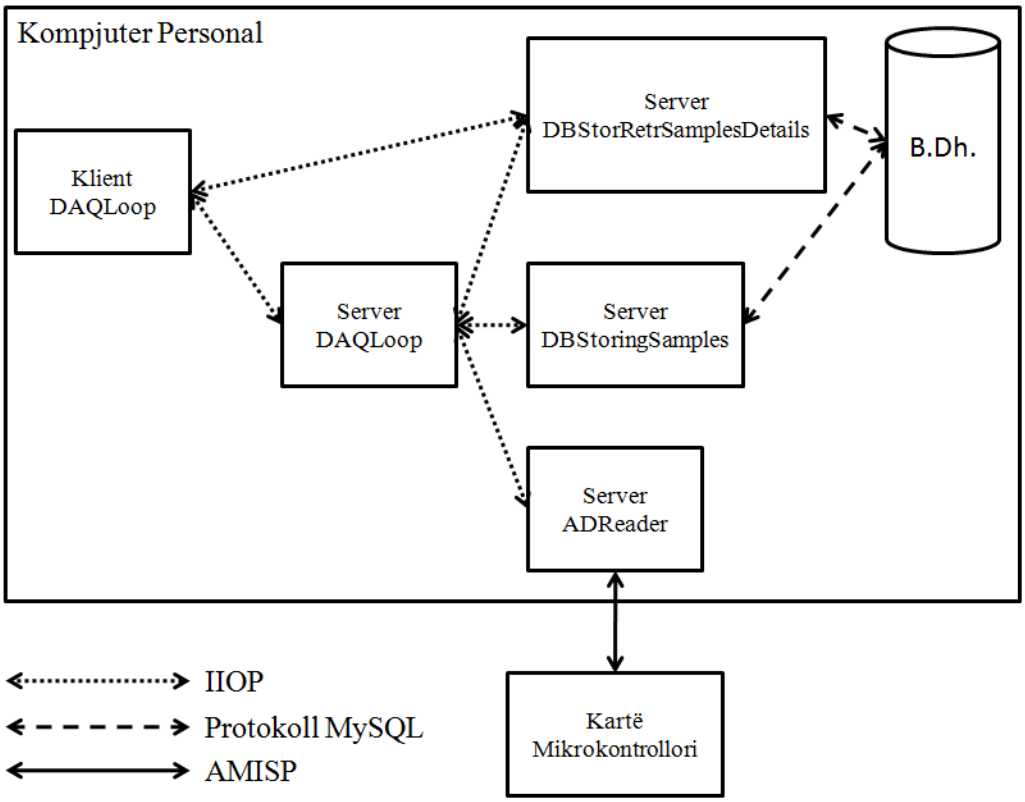


Figura 7.2 Arkitektura e sistemit të marrjes së të dhënave

Serveri DAQLoop përdoret për lexim periodik të serverit lidhës ADReader, përpunim të të dhënave dhe ruajtje të tyre në bazë të dhënash (B.Dh.). Serveri lidhës ADReader është i lidhur për serverin në mikrokontrollor që i lexon vlerat e tensionit nga konvertori A/D. Serveri DAQLoop merr instruksionet nga përdoruesi përmes klientit DAQLoop. Përdoruesi mund të fillojë, ndërpresë (pezullojë) dhe ndalë marrjen e të dhënave. Ai mund të përcaktojë transformime të të dhënave nga vlera digjitale e lexuar nga konvertori në analoge. Më tej, ai mund të ngrejë alarme kur vlerat e lexuara nga sensorët (konvertori) janë më të vogla apo më të mëdha se limitet e përcaktuara. Parametrat e të gjitha këtyre mundësive ruhen në bazë të dhënash MySQL dhe manipulohen përmes serverëve DBStorRetrSamplesDetails dhe DBStoringSamples nga klienti dhe serveri DAQLoop, ose mund t'u qasemi nga ndonjë klient tjetër të zhvilluar nga përdoruesi.

Serverët përcaktohen përmes skedarit IDL. Në shtojcat K.2 dhe K.3 janë dhënë skedarët IDL të serverit DAQLoop.

Nga operacionet e serverit DAQLoop në shtojcën K.2 janë ai `run` dhe `stop`, të paraqitur në shtojcën K.4. Operacioni `runWithPeriod` paraqet modifikim (përgjithësim) të operacionit `run`, duke e caktuar edhe periodën dhe kohën e fillimit të mostrimit për dallim nga ajo e nënkuptuar prej 5000 dhe 0 milisekondave, respektivisht.

Nga rreshtat 1 deri në 15 mund të shihet se serveri përdor tajmer për të iu qasur periodikisht serverit lidhës, përmes metodës `executeTasks` e cila ekzekutohet në periodë të caktuar. Metoda `executeTasks` fillimisht përcakton mostrën (rreshti 21) e cila merret në momentin e ezkekutimit. Më pastaj lexon 2 bajtët nga serveri lidhës (rreshti 22) duke e adresuar kanalin 0. Këtë vlerë e ruan në atributin `origBytes` të mostrës së krijuar paraprakisht. Në rreshtin 24 lexohet koha nga kompjuteri e cila ruhet në atributin përkatës të mostrës (`DateAndTime`). Në rreshtin 25 i caktohet numri identifikues i grupit të mostrave të cilës i takon mostra e tanishme. Numri identifikues i grupit caktohet për një lloj marrje të dhëne p.sh. për matjen e jetëgjatësisë së një baterie të caktuar. Në rreshtin 26 shikohet nëse mostra e marrë duhet të transformohet. Me fjalë të tjera nëse duhet të ruhet vetëm origjinali duke insertuar vlerë 0 për vlerën e vërtetë të saj (rreshtat 28 dhe 29), apo duhet të kalkulohet vlera e saj reale duke përdorur transformimin tanimë të përcaktuar (rreshti 33) dhe të ruhen në bazë së bashku me parametrat tjerë (rreshti 34). Në vijim të metodës `executeTasks` vie kodi i provës së limiteve gjegjësisht manipulimit me alarme, por që për arsye thjeshtimi nuk është paraqitur në këtë sekuencë (rreshti 37). Në fund, është paraqitur metoda stop e ndaljes së leximit, në rreshtat 46 deri në 55.

Klienti DAQLoop është i realizuar si aplikacion konzolë, që pranon komandat përmes rreshtit komandues. Disa nga komandat që mund të ekzekutohen në serverin DAQLoop përmes këtij klienti janë: krijimi i grupit të të dhënave (new details), ndërrimi i grupit të të dhënave (change details id), fillimi i marrjes së të dhënave (run [period delay]), etj.

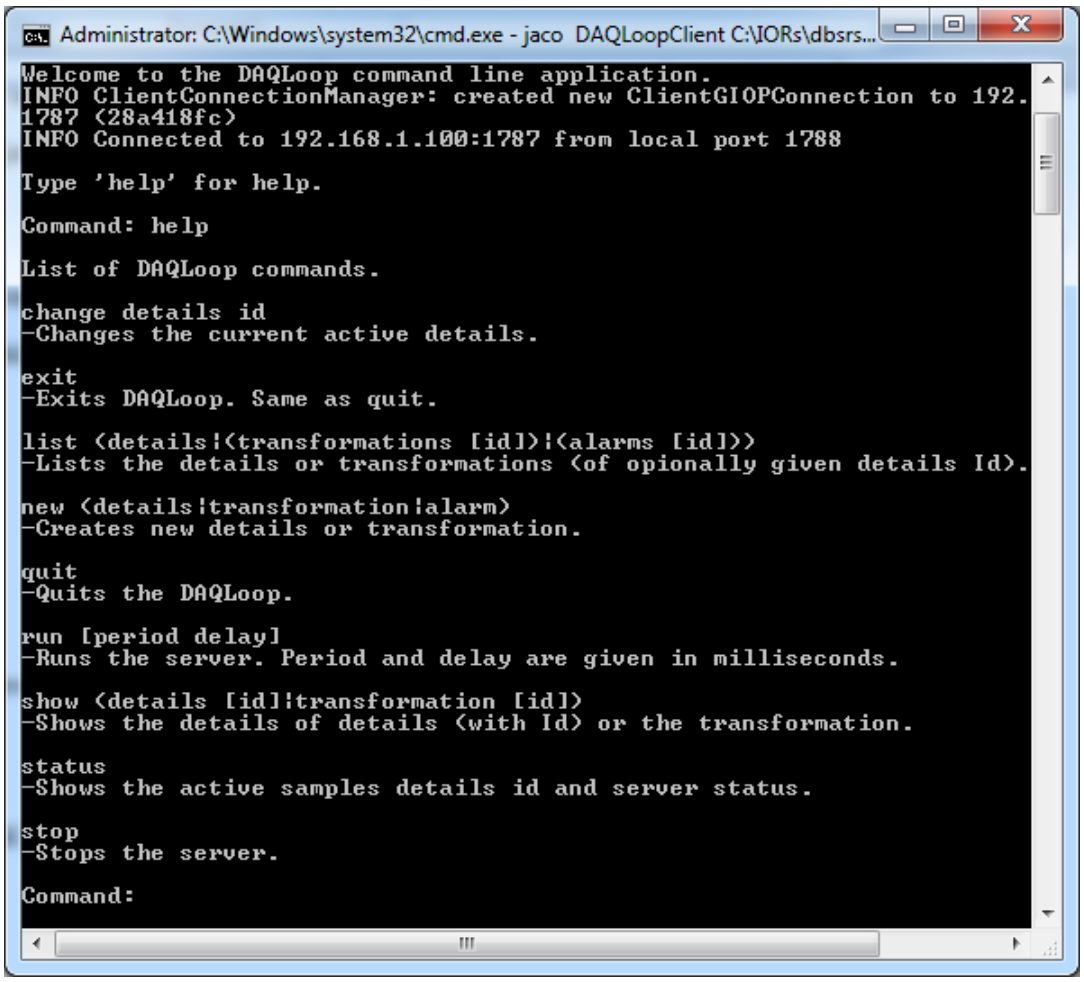


Figura 7.3 Pamje e aplikacionit konzolë të DAQLoop klientit

Të gjitha këto komanda mund të ekzekutohen në rreshtin komandues të aplikacionit, pas fjalës 'Command:', siç është paraqitur në figurën 7.3 (shih fundin e figurës). Për të përkujtuar ndonjërën nga komandat ose edhe sintaksën e thirrjes së saj, mund të jepet komanda 'help' siç kemi bërë në figurën 7.3 (shih fillimin e figurës).

7.4 Lëshimi në punë i sistemit dhe mostrat e lexuara

Hardueri i sistemit përbëhet nga laptop PC, kartë 8051 [19] dhe kartë e vogël elektronike experimentale.

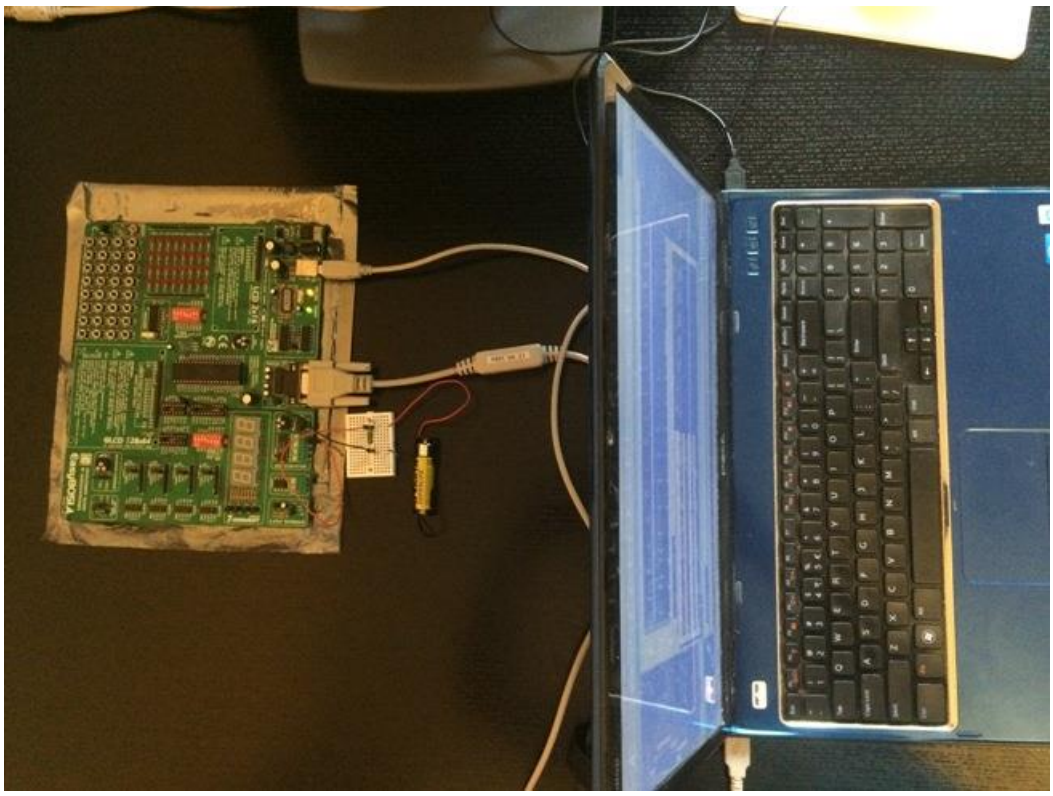


Figura 7.4 Hardueri i sistemit për marrje të dhënash

Në PC ekzekutohet softueri i organizuar si në figurën 7.2, përderisa karta 8051 është e lidhur për PC përmes kabllos USB-në-Serial për shkëmbim të të dhënave (komunikim të bazuar në AMISP) si dhe përmes një tjetër lidhjeje USB për programim të mikrokontrollorit dhe furnizim me fuqi të kartës 8051. Karta 8051 përbëhet, krahas komponenteve tjera, edhe nga konvertori analog-në-digjital (A/D) që është i lidhur për mikrokontrollor. Daljet e kartës elektronike eksperimentale janë të lidhura për hyrjet e konvertorit A/D. Pamja fizike e sistemit të përshkruar harduerik është paraqitur në figurën 7.4.

Në kartën elektronike eksperimentale është realizuar skema e thjeshtë e treguar në figurën 7.1. Ajo përbëhet prej një baterie AA 1,5 V lidhur për rezistor ohmik prej 220 Ω. Daljet (polet) e baterisë lidhen për hyrjet e konvertorit A/D, duke qenë kështu të matura periodikisht dhe përmes serverit në mikrokontrollor të përcjellura serverit lidhës, që më tej i përcillen serverit DAQLoop që i përpunon dhe i ruan në bazë të dhënash. Qëllimi i skemës është matja e shkarkimit (shpenzimit) të baterisë në rezistor gjatë kohës, duke e mostruar daljen e saj dhe duke e ruajtur atë në

bazë të dhënash së bashku me kohën e mostrimit. Perioda e mostrimit në matjet tona është 20 sekonda. Janë marrë gjithsejt 2095 mostra, duke zgjatur kështu matja për afro 11,64 orë (2095 x 20 sekonda). Disa nga mostrat janë paraqitur në figurën 7.5 ashtu siç duken në bazën e të dhënave MySQL.

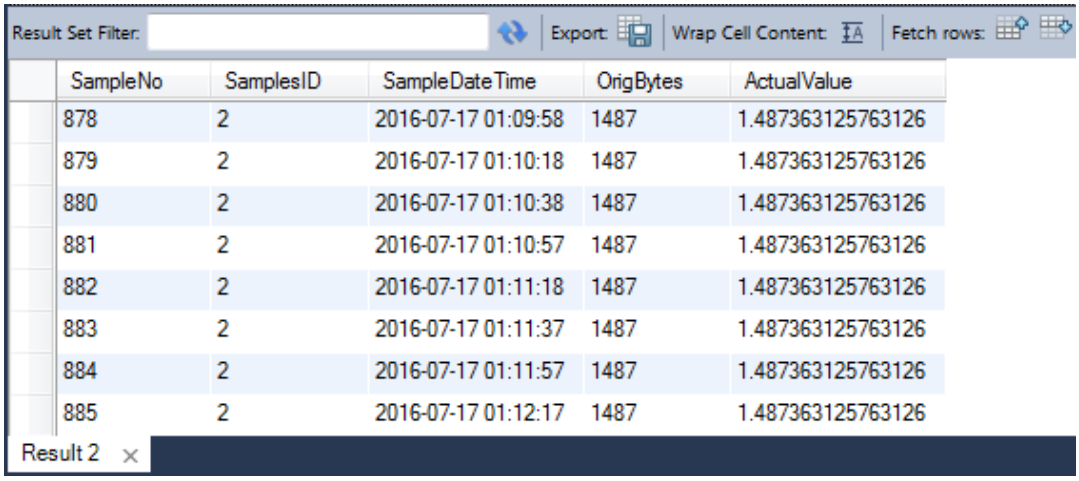


Figura 7.5 Disa nga mostrat e marra në bazën e të dhënave MySQL

Nga figura 7.5 mund të kuptohet se kolonat e tabelës ku ruhen mostrat janë të njëjta me atributet e strukturës Sample në shtojcën K.3. Mbi kuptimin i tyre mund të përfundohet lehtë nga emërtimi. Kështu, kolona 'SampleNo' përmban numrin identifikues të mostrës së marrë. Kolona 'SamplesID' përmban numrin identifikues të grupit të mostrave, që është unik për mostrat e matjes e caktuar. Pra, e identifikon grupin e mostrave. Kolona 'SampleDateTime' përmban datën dhe kohën kur merret mostra e caktuar. Kolona 'OrigBytes' përmban vlerën origjinale të bajtëve të lexuar nga sensori, në këtë rast nga konvertori A/D. E fundit, kolona 'ActualValue' përmban vlerën analoge të kalkuluar në bazë të bajtëve nga kolona 'OrigBytes' dhe transformimit të përcaktuar për matjen e dhënë (shih shtojcën K.4 rreshti 33 ose rreshti 28).

Për mostrat e marrura, e kemi bërë edhe paraqitjen grafike të tensionit të matur (figura 7.5 kolona 'ActualValue') në funksion të numrit të mostrës (figura 7.5 kolona 'SampleNo'). Tensionin e kemi paraqitur në boshtin vertikal, të shprehur ne V, ndërsa numrin e mostrave në boshtin horizontal. Numri i mostrave e ka kuptimin edhe të kohës, pasi koha e mostrimit mund të llogaritet si *numri i mostrës x perioda e*

mostrimit (perioda në këtë rast është 20 sekonda). Grafi i tillë është paraqitur në figurën 7.6.

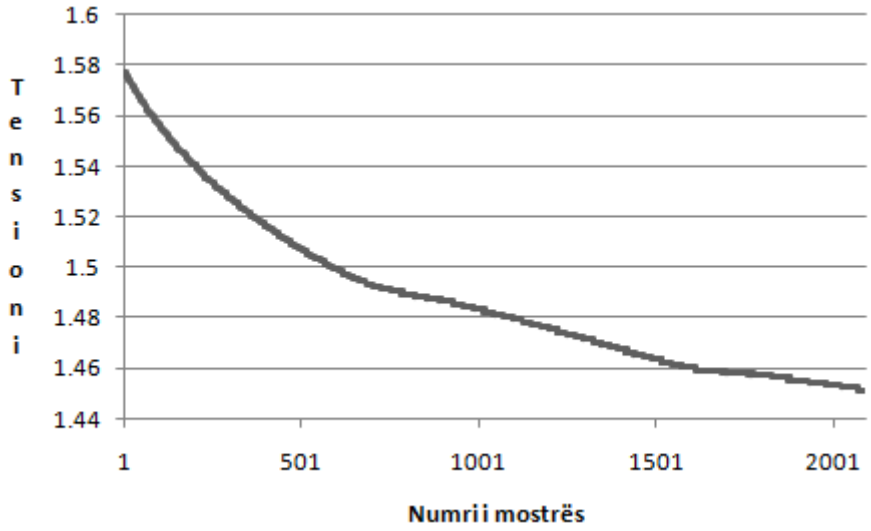


Figura 7.6 Tensioni i baterisë në funksion të numrit të mostrës (kohës)

KAPITULLI 8

Përfundime dhe puna për të ardhmen

8.1 Përfundime

Me futjen e konceptit të serverit lidhës është bërë e mundur që Arkitekturat e Orientuara në Shërbime (AOSh), siç janë CORBA dhe Shërbimet Ueb, të implementohen edhe në pajisje shumë të kufizuara siç janë mikrokontrollorët 8051 [5], me vetëm 12 kilobajtë kujtesë programuese dhe 128 bajtë kujtesë RAM. Klienti e ka përshtypjen që operacionet ekzekutohen në serverin lidhës me funksionalitet të plotë, duke përfituar kështu nga funksionalitetet që teknologjia AOSh përkatëse i ofron. Të dy, serveri lidhës dhe ai i mbyllur gjenerohen nga gjeneratori i kodit që e kemi zhvilluar. Gjeneratori i kodit është 100 % *retargetable*, që do të thotë se nga listat e objekteve të ndërtuara me parsimin e skedarëve të dhënë IDL (ang. Interface Definition Language) dhe atij implementues, mund të gjenerohet kod për serverin lidhës të cilësdo platformë dhe server i mbyllur për çfarëdo lloji të pajisjes. Kjo është bërë e mundur, duke ndarë plotësisht nga njëra tjetra fazat e ndërtimit të listës së objekteve dhe gjenerimit të kodit. Duhet ri-theksuar se nëse teknologjia e re AOSh përkrah zhvillimin në gjuhën programuese Java, atëherë me modifikim të vogël të gjeneratorëve të serverëve lidhës JacORB ose JAX-WS që i kemi paraqitur në këtë punim, mund të ndërtohet gjeneratori i serverit lidhës për teknologjinë e re.

Gjenerimi i kodit dhe shkëmbimi i mesazheve është i optimizuar, duke i ndërtuar fushat e mesazhit në mënyre adaptive, bazuar në protokollin AMISP (ang. Adaptive Minimal Inter Server Protocol). Kështu, në mesazh janë prezente vetëm ato fusha që e identifikojnë operacionin në mënyrë unike. Fushat janë bajt i plotë duke i bërë të përshtatshme për përpunim në mikrokontrollorët 8 bitësh, pasi instruksionet e mikrokontrollorit bazohen në bajt (8 bitë). Kjo rezulton në madhësi shumë të vogla reale të kodit në skedarët HEX, të fituar pas kompilimit të atyre të gjeneruar në asembler. Kjo paraqet përparësi në krahasim me qasjet në literaturë që përdorin

protokollin EXI (ang. Efficient Xml Interchange) që është i bazuar në komprimim/dekomprimim, që e rrit dukshëm kodin e serverit të pajisjes së mbyllur. Edhe pse përdoret vetëm një bajt për identifikim të opsionit (ndërtimin e fushës), kjo nuk paraqet mangësi pasi 256 opsione të mundshme që mund të kodohen me atë rast paraqesin numër mëse të mjaftueshëm. Sa i përket parametrave të operacioneve, përkrahen të gjithë tipat numerik të plotë, ata me pikë të lëvizshme dhe të tjerët me madhësi fikse nga ata të përcaktuar me IDL të funksionalitetit të plotë.

Në kapitullin 6 kemi parë shembuj të madhësisë së kodit të gjeneruar dhe mesazheve korresponduese të komunikimit ndërmjet serverit lidhës dhe atij në pajisje të mbyllur. Mesazhet janë paraqitur të detajuara, duke e vërtetuar gjenerimin e vetëm fushave të domosdoshme. Madhësitë e kodit pas kompilimit pra madhësia e reale e kodit në skedarin HEX është prej vetëm qindra bajtësh, duke qenë më i vogël (mirë) se ai në literaturën relevante që gjenerojnë madhësi të rendit të dhjetëra deri në disa qindra kilobajtë. Madhësia e mesazheve prej disa bajtësh, është e krahasueshme me ato në literaturën relevante. Por, kjo nuk paraqet mangësi pasi mesazhet e gjeneratorit tonë krahasohen me ato të komprimuara të literaturës përkatëse. Poashtu, kemi bërë matje të kohëve të komunikimit klient/server i mbyllur dhe i kemi krahasuar me ato në literaturën relevante. Edhe në këtë rast kohët tona kanë qenë disa dhjetëra herë më të vogla se ato në literaturën relevante – që do të thotë se komunikimi ka qenë më i shpejtë, që paraqet rezultat të rëndësishëm.

Në fund, në kapitullin 7 kemi paraqitur një sistem për marrje të dhënash. Ai paraqet sistem për monitorim të jetëgjatësisë së baterive. Përveç jetësimit të konceptit të zhvilluar në këtë tezë, pra atij server lidhës – server në mikrokontrollor, aty kemi prezantuar edhe një platformë homogjene të bazuar në JacORB. Platforma lehtëson punën e leximit nga sensorët në mënyrë periodike, kalkulimin e mostrave, ruajtjen e tyre në bazë dhe ngrerjen e alarmeve në rast të tejkalimit të vlerave kufitare të paracaktuara. E kemi parë poashtu se janë lexuar një numër i madh mostrash, që demonstron se sistemi ka qenë i qëndrueshëm duke punuar pa ndërprerë një kohë të gjatë.

8.2 Puna për të ardhmen

Puna për të ardhmen do të duhej të orientohej në dy drejtime:

Në implementimin e sigurisë së komunikimit ndërmjet serverit në mikrokontrollor dhe atij lidhës, dhe

Në shtimin e funksionaliteteve, respektivisht zgjerimin e gramatikës me deklarime shtesë nga ajo me funksionalitet të plotë të gramatikës IDL

1. Nga gramatika aktuale e skedarit IDL mund të kuptohet se ndërfaqja mund të deklarohet që e zgjeron atë `ISEASecured`. Kjo, është bërë me qëllim që ndërkohë ose në të ardhmen, si alternativë, komunikimi ndërmjet serverit lidhës dhe atij të mbyllur të bëhet i enkriptuar (i siguruar). Arsyet janë të shumëta, dhe një nga to është se përcaktimi aktual i protokollit AMISP, për shkak të gjenerimit minimal të fushave, lë mundësi të involvimit të palës së tretë për ta kompromet uar sistemin, duke gjeneruar p.sh. kërkesa që nuk e kanë origjinën nga serveri lidhës për të cilin është i destinuar serveri i mbyllur. Edhe pse serveri lidhës dhe ai i mbyllur aktualisht komunikojnë përmes porteve seriale të realizuar përmes kabllove përkatës fizik, në të ardhmen ata do të mund të komunikonin edhe pa tela që do ta rriste edhe më tepër ekpozimin ndaj faktorëve të jashtëm të padëshiruar. Në përkufizimin e ndërfaqes ISEASecured fjala SEA qëndron për Scalable Encryption Algorithm [47] – një algoritëm i destinuar për pajisje të mbyllura. Tanimë ne i kemi bërë disa implementime të publikuara në [48], ku duhet të theksohet se në komunikim krahas fushave të parapara me protokollin AMISP, është shfaqur nevoja që të shtohen edhe fusha të tjera për ta përcaktuar kërkesën dhe inicuesin në mënyrë unike. Të rikujtojmë se komunikimi i sigurt në mes të klientit dhe serverit lidhës bëhet përmes mekanizmave të sigurisë që teknologjia përkatëse e implementimit të tyre i ofron.

2. Është e qartë se gramatika që aktualisht realizohet nga gjeneratori i kodit të prezantuar në këtë tezë, është një nënbashkësi e asaj me funksionalitet të plotë që përcakton tërë gjuhën IDL [31]. Zgjerimi eventual i gramatikës drejt asaj me funksionalitet të plotë do të kishte pasoja. Para së gjithash do të ndikonte në rritje drastike të madhësisë së kodit dhe mesazheve. Nga ana tjetër, implementimi në

asembler i serverit të mbyllur do të paraqiste vështirësi për shkak të detajeve të shumta që do ti kishte për barrë zhvilluesi i shërbimit. Kështu, do të ishte e përshtatshme që gjenerimi respektivisht zhvillimi të bëhej në ndonjë gjuhë të lartë programuese, për të cilën ekziston komplatori përkatës për pajisjen e mbyllur që e përdorim, në këtë rast mikrokontrollrin 8051. Mundësisht kompilatori do të duhej të ishte pa pagesë, siç është p.sh. SDCC (ang. Small Device C Compiler) [49], që është edhe me kod të hapur.

REFERENCAT

[1] H. Mössenböck, "The Compiler Generator Coco/R - User Manual," Johannes Kepler University Linz, Institute of System Software, Linz, 2010, e aksesueshme në faqen http://www.ssw.uni-linz.ac.at/coco/, aksesuar së fundmi më 25 Janar 2018

[2] N. Wirth, “What Can We Do about the Unnecessary Diversity of Notation for Syntactic Definitions?”, Communications of the ACM, November 1977

[3] I. Scott MacKenzie, "The 8051 Microcontroller", Prentice Hall, Third Edition, 1995

[4] M. Verle, "Architecture and Programming of 8051 Microcontrollers", Mikroelektronika, 2008, e aksesueshme në faqen https://www.mikroe.com/ebooks/architecture-and-programming-of-8051-mcus/introduction, aksesuar së fundmi më 25 Janar 2018

[5] Atmel Corporation, "AT89S8253 Datasheet", 2008

[6] “8051 Addressing modes”, e aksesueshme në faqen http://www.circuitstoday.com/8051-addressing-modes, aksesuar së fundmi më 25 Janar 2018

[7] “8051 Tutorial: Serial Communication”, e aksesueshme në faqen http://www.8052.com/tutser.phtml, aksesuar së fundmi më 25 Janar 2018

[8] “The 8051, Serial Communication”, e aksesueshme në faqen http://www.edsim51.com/8051Notes/8051/serial.html, aksesuar së fundmi më 25 Janar 2018

[9] “8051 Instruction Set Manual: Instructions”, e aksesueshme në faqen http://www.keil.com/support/man/docs/is51/is51_instructions.htm, aksesuar së fundmi më 25 Janar 2018

[10] “8051 Instruction Set”, e aksesueshme në faqen http://www.win.tue.nl/~aeb/comp/8051/set8051.html, aksesuar së fundmi më 25 Janar 2018

[11] Y. Nakamoto, K. Yabuuchim, T. Osaki, T. Kishida, T. Hara, I. Abe, A. Kitamura, “Proposing A Hybrid Software Execution Environment for Distributed Embedded Systems”, 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, 2010

[12] A. Sleman, R. Moeller, “Micro SOA Model for Managing and Integrating Wireless Sensor Network Into IP-Based Networks”, 2010 Second International Conference on Computational Intelligence, Communication Systems and Networks, 2010

[13] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, K. Kim, “TinyREST – a Protocol for Integrating Sensor Networks into the Internet”, in Proc. of REALWSN, pp. 101-105, 2005

[14] R. van Engelen, K. Gallivan, "The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks", 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), 2002, e aksesueshme në faqen https://www.cs.fsu.edu/~engelen/ccgrid.pdf, aksesuar së fundmi më 25 Janar 2018

[15] A. Bagnasco, D. Cipolla, D. Occhipinti, A. Preziosi, A. M. Scapolla, "Application of web services to heterogeneous networks of small devices", Proceedings of the 8th WSEAS Int. Conference on Automatic Control, Modeling and Simulation, Prague, Czech Republic, 2006

[16] G. Moritz, S. Prueter, D. Timmermann, F. Golatowski, "Deployment of embedded web services in real-time systems", Scalable Computing: Practice and Experience, Vol. 10, No. 3, pp.265-275, 2009

[17] T. R. Szarek, "On the use of microcontrollers for data acquisition in an introductory measurements course", Master's Thesis, University of Notre Dame, Indiana, 2003

[18] S. R. Saraf, R. M. Holmukhe, "Microcontroller based data acquisition system for electrical motor vibrations using VB software", Indian Journal of Computer Science and Engineering (IJCSE), 2011

[19] Easy8051 v6 Development System, e aksesueshme në faqen https://www.mikroe.com/easy8051, aksesuar së fundmi më 25 Janar 2018

[20] BIG8051 Development System, e aksesueshme në faqen https://www.mikroe.com/big8051, aksesuar së fundmi më 25 Janar 2018

[21] U. Brinkschulte, A. Bechina, F. Picioroaga, E. Schneider, "Distributed Real-Time Computing for Microcontrollers - the OSA+ Approach", Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '02), 2002

[22] W3C Working Group Note, Web Services Architecture, 2004, e aksesueshme në faqen http://www.w3.org/TR/ws-arch/, aksesuar së fundmi më 25 Janar 2018

[23] OMG's CORBA Web-site, e aksesueshme në faqen http://www.corba.org/, aksesuar së fundmi më 25 Janar 2018

[24] .NET Remoting Overview, e aksesueshme në faqen http://msdn.microsoft.com/en-us/library/kwdt6w2k(v=vs.71).aspx, aksesuar së fundmi më 25 Janar 2018

[25] Chervet A., Considerations for Using XML Web Services for Device-to-device Communication, Echelon

[26] C51ASM, e aksesueshme në faqen http://www.atmel.com/tools/C51ASM.aspx, aksesuar së fundmi më 15 Korrik 2016

[27] Schneider J., Kamiya T., "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", W3C Recommendation 11 February 2014, e aksesueshme në faqen http://www.w3.org/TR/exi/, aksesuar së fundmi më 25 Janar 2018

[28] IETF, "Constrained Application Protocol (CoAP)", 2012, e aksesueshme në faqen https://datatracker.ietf.org/doc/draft-ietf-core-coap/, aksesuar së fundmi më 15 Korrik 2016

[29] W3C Working Group Note, Web Services Glossary, 2004, e aksesueshme në faqen http://www.w3.org/TR/ws-gloss/, aksesuar së fundmi më 25 Janar 2018

[30] JacORB, e aksesueshme në faqen www.jacorb.org/, aksesuar së fundmi më 25 Janar 2018

[31] Common Object Request Broker Architecture (CORBA) Specification, Version 3.3, Part 1: CORBA Interfaces, e aksesueshme në faqen http://www.omg.org/spec/CORBA/3.3/Interfaces/PDF, aksesuar së fundmi më 25 Janar 2018

[32] Wöß A., Löberbauer M., Mössenböck H., Compiler generation tools for C#, IEE Proc.-Softw., Vol. 150, No. 5, October 2003

[33] M. Henning, S. Vinoski, "Advanced CORBA Programming with C++", Addison Wesley, First Edition, 1999

[34] F. Bolton, "PURE CORBA", Sams Publishing, 2002

[35] Object Management Group, e aksesueshme në faqen http://www.omg.org/, aksesuar së fundmi më 25 Janar 2018

[36] G. Brose, A. Vogel, K. Duddy, "Java Programming with CORBA - Advanced Techniques for Building Distributed Applications", John Wiley & Sons, Third Edition, 2001

[37] S. Ahmed, "CORBA Programming Unleashed", Sams Publishing, First Edition, 1998

[38] The JacORB Team, "JacORB 3.4 Programming Guide", January 15, 2014, e aksesueshme në faqen http://www.jacorb.org/releases/3.4/ProgrammingGuide.pdf, aksesuar së fundmi më 25 Janar 2018

[39] Java IDL: IDL to Java Language Mapping, e aksesueshme në faqen http://docs.oracle.com/javase/7/docs/technotes/guides/idl/mapping/jidlMapping.html, aksesuar së fundmi më 25 Janar 2018

[40] S. Käbisch, D. Peintner, J. Heuer, H. Kosch, "XML-based Web Service Generation for Microcontroller-based Sensor Actor Networks", 8th IEEE International Workshop on Factory Communication Systems (WFCS), 2010

[41] S. Käbisch, D. Peintner, J. Heuer, H. Kosch, "Efficient and Flexible XML-based Data-Exchange in Microcontroller-based Sensor Actor Networks", IEEE 24th

International Conference on Advanced Information Networking and Applications Workshops, 2010

[42] S. Käbisch, D. Peintner, J. Heuer, H. Kosch, “Optimized XML-based Web Service Generation for Service Communication in Restricted Embedded Environments”, IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA), 2011

[43] Modbus Organization, “Modbus Protocol”, e aksesueshme në faqen http://www.modbus.org/specs.php, aksesuar së fundmi më 25 Janar 2018

[44] Modbus Organization, “MODBUS over Serial Line”, Specification and Implementation Guide V1.02, 2006, e aksesueshme në faqen http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf, aksesuar së fundmi më 25 Janar 2018

[45] The gSOAP toolkit fact sheet - version release 2.8.12 and higher, e aksesueshme në faqen https://www.cs.fsu.edu/~engelen/factsheet.pdf, aksesuar së fundmi më 25 Janar 2018

[46] Microchip, MCP3204/3208 Datasheet, e aksesueshme në faqen http://ww1.microchip.com/downloads/en/DeviceDoc/21298c.pdf, aksesuar së fundmi më 25 Janar 2018

[47] F.X. Standaert, G. Piret, N. Gershenfeld, J. J. Quisquater, "SEA a Scalable Encryption Algorithm for Small Embedded Applications", Workshop on RFIP and Lightweight Crypto, 2005

[48] K. Hyseni, N. Frasheri, "Security in embedded sensor actor networks", International Conference on Information Systems and Technology Innovations - ISTI'2014, Tirana 6-7 June 2014

[49] "SDCC - Small Device C Compiler", e aksesueshme në faqen http://sdcc.sourceforge.net/, aksesuar së fundmi më 25 Janar 2018

[50] K. Hyseni, N. Frasheri, “Microcontrollers as components of service oriented architectures”, 3rd IEEE Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2014

[51] K. Hyseni, N. Frasheri, “Bind server – an adapter from high level to embedded communication in service oriented architectures”, 9th Annual South-East European Doctoral Student Conference (9th SEE DSC), Thessaloniki, Greece, 2014

[52] K. Hyseni, N. Frasheri, “Integrating embedded devies in SOA through optimized re-targetable code generation”, International Journal on Information Technologies & Security, Vol. 8 Issue 3, p3-28, 26p, 2016

[53] K. Hyseni, N. Frasheri, “Low cost SOA based platform independent back-end data acquisition system”, International Journal on Information Technologies & Security, Vol. 8 Issue 3, p53-64, 12p, 2016

[54] K. Hyseni, N. Frasheri, “Code and message footprints of generated code for embedded devices as components of SOA”, International Journal of Science, Innovation and New Technology, Vol. 1 Issue 16, p37-43, 7p, 2016

LISTA E BOTIMEVE

Konferenca:

1. **K. Hyseni,** N. Frasheri, "Microcontrollers as components of service oriented architectures", 3rd IEEE Mediterranean Conference on Embedded Computing (MECO), 2014, Budva, Montenegro
2. **K. Hyseni,** N. Frasheri, "Bind server – an adapter from high level to embedded communication in service oriented architectures", 9th Annual South-East European Doctoral Student Conference (9th SEE DSC), 2014, Thessaloniki, Greece
3. **K. Hyseni,** N. Frasheri, "Security in embedded sensor actor networks", 5th International Conference – Information Systems and Technology Innovation: projecting trends in New Economy, 2014, Tiranë, Albania

Revista:

1. **K. Hyseni,** N. Frasheri. Integrating embedded devies in SOA through optimized re-targetable code generation. International Journal on Information Technologies & Security. 2016, Vol. 8 Issue 3, p3-28. 26p
2. **K. Hyseni,** N. Frasheri. Low cost SOA based platform independent back-end data acquisition system. International Journal on Information Technologies & Security. 2016, Vol. 8 Issue 3, p53-64. 12p
3. **K. Hyseni,** N. Frasheri. Code and message footprints of generated code for embedded devices as components of SOA. International Journal of Science, Innovation and New Technology. 2016, Issue June, 8p

SHTOJCA A

Kodi i serverit në mikrokontrollor

A.1 Kodi i bllokut 1 në Asembler

```
ORG 0000H
JMP InitSerComAndEnblSerInt
```

A.2 Kodi i bllokut 2 në Asembler

```
ORG 0023H ; Serial port ISR vector address
JMP SPISR
```

A.3 Kodi i bllokut 3 në Asembler

```
ORG 0030H
InitSerComAndEnblSerInt:
MOV SCON, #50H
MOV TMOD, #20H
MOV TH1, #239 ; Set 1200 baud rate
SETB TR1 ; Start timer 1
SETB PS ; Set higher level (1) priority for serial port interrupt
SETB ES ; Enable serial port interrupt
SETB EA ; Enable global interrupt
```

A.4 Kodi i bllokut 4 në Asembler

```
MAIN:
JMP $ ; do nothing, just wait for incomming requests
```

A.5 Kodi i bllokut 5 në Asembler

```
SPISR:
CLR ES ; Disable serial interrupt, continue receiving request …

JNB RI, $    ; Read the first byte
CLR RI       ; i.e. message length
MOV A, SBUF

MOV R0, #144    ; Message length placed
MOV @R0, A  ; to RAM location 144

DEC A

CJNE A, #0, ContReadStreamBytes
JMP ClrRenAndProc
ContReadStreamBytes:
MOV R1, A

RecvIncStreamBytes:
INC R0
JNB RI, $
CLR RI
MOV @R0, SBUF
DJNZ R1, RecvIncStreamBytes

ClrRenAndProc:
CLR REN ; Disable receiving, process incoming stream and send …
```

A.6 Kodi i bllokut 7 në Asembler

```
; Read interface ID from RAM location 145
MOV R0, #145
MOV A, @R0

ChckIntID_TempSensor_0:
CJNE A, #0, ChckIntID_TempSensor_0_Failed ; Check interface ID
JMP Chck_TempSensor_0_operationIDs
ChckIntID_TempSensor_0_Failed:
JMP ChckIntID_Termostat_1
```

A.7 Kodi i bllokut 9 në Asembler

```
; Read operation ID from RAM location 146
Chck_TempSensor_0_operationIDs:
MOV R0, #146
MOV A, @R0
Chk_IntImpl_TempSensor_0_ATTR__get_minTempRead_ID:
```

```
6 CJNE A, #0,
Chk_IntImpl_TempSensor_0_ATTR__get_minTempRead_ID_Failed ; Check
operation ID
7 JMP IntImpl_TempSensor_0_ATTR__get_minTempRead_ID
8 Chk_IntImpl_TempSensor_0_ATTR__get_minTempRead_ID_Failed:
9 JMP Chk_IntImpl_TempSensor_0_ATTR__get_maxTempRead_ID
```

A.8 Blloku 10 i komenteve në Asembler

```
; Please find in and inout parameters at RAM locations as follows:
; parameter, name: devID, direction: in, CORBA type: short, size: 2,
RAM locations: 147..148
; Please place return, out and inout parameters at RAM locations as
follows:
; parameter, direction: return, CORBA type: float, size: 4, RAM

    locations: 144..147
```

A.9 Blloku 11 i komenteve në Asembler

```
; Uncomment the following code to terminate sucessfully at the end of
your code
; MOV A, #0
; JMP DcdRspns_IntImpl_TempSensor_0_STOP_readTemp
; In case the operation terminates with DeviceNotFound exception ...
; Please place param(s) at following RAM addresses:
; parameter, name: deviceID, size: 2, RAM locations: 144..145
; Uncomment the following code to terminate with exception
DeviceNotFound (ID: 1) at the end of your code
; MOV A, #1
; JMP DcdRspns_IntImpl_TempSensor_0_STOP_readTemp
```

A.10 Blloku 12 i gjenerimit të kodit për lexim të atributit në Asembler

```
1 MOV R0, #128
2 MOV R1, #144
3 MOV R2, #4
4 IntImpl_TempSensor_0_ATTR__get_minTempRead_bytes_loop:
5 MOV A, @R0
6 MOV @R1, A
7 INC R0
8 INC R1
9 DJNZ R2, IntImpl_TempSensor_0_ATTR__get_minTempRead_bytes_loop
```

A.11 Blloku 13 i përgjigjes në Asembler

```
 1 DcdRspns_IntImpl_TempSensor_0_STOP_readTemp:
 2 DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0:
 3 CJNE A, #0,
DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0_Failed
 4 JMP DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0_RSPNS
 5 DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0_Failed:
 6 JMP DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_1
 7 DcdRspns__IntImpl_TempSensor_0_STOP_readTemp_STTS_0_RSPNS:
 8 ; Send response header for status OK (0)
 9 MOV R0, #4
10 MOV DPTR, #DB_IntImpl_TempSensor_0_STOP_readTemp_STTS_0_resp_head
11 CALL SendResponseHeader
12 ; Send response body (return, out and inout parameters) from RAM
13 MOV R1, #4
14 CALL SendResponseBodyRAM
15 JMP EnSerComAndInt ; Jump to restore point
```

A.12 Blloku 14 i riinicializimit në Asembler

```
1 EnSerComAndInt:
2 SETB REN
3 SETB ES
4 RETI
```

A.13 Blloku 15 i nënprogramit të *header*-it të përgjigjes në Asembler

```
 1 SendResponseHeader:
 2 SndByteRH:
 3 CLR A
 4 MOVC A, @A+DPTR
 5 MOV SBUF, A
 6 JNB TI, $
 7 CLR TI
 8 INC DPTR
 9 DJNZ R0, SndByteRH
10 RET
```

A.14 Blloku 16 i nënprogramit të trupit të përgjigjes në Asembler

```
SendResponseBodyRAM:
MOV R0, #144
SendResponseBodyByte:
MOV A, @R0
MOV SBUF, A
JNB TI, $
CLR TI
INC R0
DJNZ R1, SendResponseBodyByte
RET
```

A.15 Blloku 17 i bajtëve të *header*-ëve të përgjigjeve në Asembler

```
DB_IntImpl_TempSensor_0_ATTR__get_minTempRead_resp_head: DB 7, 0, 0
DB_IntImpl_TempSensor_0_ATTR__get_maxTempRead_resp_head: DB 7, 0, 1
DB_IntImpl_TempSensor_0_STOP_readTemp_STTS_0_resp_head: DB 8, 0, 2,
0
DB_IntImpl_TempSensor_0_STOP_readTemp_STTS_1_resp_head: DB 6, 0, 2,
1
DB_IntImpl_Termostat_1_ATTR__get_minTempAllowed_resp_head: DB 7, 1,
0
DB_IntImpl_Termostat_1_ATTR__get_maxTempAllowed_resp_head: DB 7, 1,
1
DB_IntImpl_Termostat_1_STOP_setTemperature_STTS_0_resp_head: DB 4,
1, 2, 0
DB_IntImpl_Termostat_1_STOP_setTemperature_STTS_1_resp_head: DB 6,
1, 2, 1
DB_IntImpl_Termostat_1_STOP_setTemperature_STTS_129_resp_head: DB
10, 1, 2, 129
```

SHTOJCA B

Kodi i serverit lidhës për JacORB

B.1 Kodi i serverit lidhës për CORBA i gjeneruar nga ndërfaqja Termostat

```
  1 class TermostatImpl1 extends TermostatPOA
  2 {
  3    MISPSerialManaged _ms;
  4    byte _interfaceID = (byte) 1;
  5    ConnectionPolicy _connPolicy =
ConnectionPolicy.CONNECTONDEMAND;
  6
  7    ...
  8
  9    public TermostatImpl1(String comPort, int baudRate,
ConnectionPolicy connPolicy)
 10    {
 11       try
 12       {
 13          _ms = new MISPSerialManaged(comPort, baudRate);
 14          this._connPolicy = connPolicy;
 15       }
 16       catch(Exception _xp)
 17       {
 18          System.out.println(_xp);
 19          System.exit(0);
 20       }
 21    }
 22
 23    public synchronized int minTempAllowed()
 24    {
 25       int _retVal = 0;
 26       byte _operationID = (byte) 0;
 27       byte _osSize = (byte) 3;
 28       byte _isSize = (byte) 7;
 29
 30       if (!_ms.isConnected)
 31          if(!_ms.connect())
 32          {
 33             System.out.println("Could not connected to embedded
server, returning default value.");
 34             return _retVal;
 35          }
 36
 37       try
 38       {
```

```
 39          StreamOperations2 _outStream = new
StreamOperations2(_osSize);
 40          _outStream.writeByte(_interfaceID);
 41          _outStream.writeByte(_operationID);
 42          _ms.isDataReady = false;
 43          _ms.outputStream.write(_outStream.getStream(), 0,
_outStream.getStreamLength());
 44          while(!_ms.isDataReady);
 45          StreamOperations2 _inStream = new
StreamOperations2(_ms.readStream);
 46          byte _readStreamLength = _inStream.readByte();
 47          byte _readInterfaceID = _inStream.readByte();
 48          byte _readOperationID = _inStream.readByte();
 49          if ((_readStreamLength == _isSize) && (_readInterfaceID
== _interfaceID) && (_readOperationID == _operationID))
 50          {
 51             _retVal = _inStream.readInt();
 52          }
 53          else
 54          {
 55             System.out.println("Wrong stream received: received
length " + _readStreamLength + "; expected interface ID " +
_interfaceID + ", received interface ID " + _readInterfaceID + ";
expected operation ID " + _operationID + ", received operation ID " +
_readOperationID + ";");
 56          }
 57          _ms.isDataReady = false;
 58       }
 59       catch(Exception _xp)
 60       {
 61          System.out.println(_xp);
 62       }
 63
 64       if ((_connPolicy == ConnectionPolicy.CONNECTONDEMAND) ||
(_connPolicy == ConnectionPolicy.ADAPTIVECONNECTIVITY))
_ms.disconnect();
 65
 66       return _retVal;
 67    }
 68
 69    ...
 70
 71    public synchronized void setTemperature(short devID, float
temp) throws DeviceNotFound, TermostatPackage.TempNotAllowed
 72    {
 73       byte _operationID = (byte) 2;
 74       byte _osSize = (byte) 9;
 75       byte _isSize = (byte) 4;
 76
 77       if (!_ms.isConnected)
 78          if(!_ms.connect())
 79          {
 80             System.out.println("Could not connected to embedded
server, returning default value.");
 81             return;
 82          }
 83
 84       try
```

```
 85        {
 86           StreamOperations2 _outStream = new
StreamOperations2(_osSize);
 87           _outStream.writeByte(_interfaceID);
 88           _outStream.writeByte(_operationID);
 89           _outStream.writeShort(devID);
 90           _outStream.writeFloat(temp);
 91           _ms.isDataReady = false;
 92           _ms.outputStream.write(_outStream.getStream(), 0,
_outStream.getStreamLength());
 93           while(!_ms.isDataReady);
 94        }
 95        catch(Exception _xp)
 96        {
 97           System.out.println(_xp);
 98        }
 99
100        _ms.isDataReady = false;
101
102        if ((_connPolicy == ConnectionPolicy.CONNECTONDEMAND) ||
(_connPolicy == ConnectionPolicy.ADAPTIVECONNECTIVITY))
_ms.disconnect();
103
104        StreamOperations2 _inStream = new
StreamOperations2(_ms.readStream);
105        byte _readStreamLength = _inStream.readByte();
106        byte _readInterfaceID = _inStream.readByte();
107        byte _readOperationID = _inStream.readByte();
108        byte _readStatusID = _inStream.readByte();
109        if ((_readStreamLength == _isSize) && (_readInterfaceID ==
_interfaceID) && (_readOperationID == _operationID) && (_readStatusID
== (byte) 0));
110        else if ((_readStreamLength == (byte) 6) &&
(_readInterfaceID == _interfaceID) && (_readOperationID ==
_operationID) && (_readStatusID == (byte) 1))
111           throw new DeviceNotFound(_inStream.readShort());
112        else if ((_readStreamLength == (byte) 10) &&
(_readInterfaceID == _interfaceID) && (_readOperationID ==
_operationID) && (_readStatusID == (byte) 129))
113           throw new
TermostatPackage.TempNotAllowed(_inStream.readShort(),
_inStream.readFloat());
114        else
115        {
116           System.out.println("Wrong stream received: received
length " + _readStreamLength + "; expected interface ID " +
_interfaceID + ", received interface ID " + _readInterfaceID + ";
expected operation ID " + _operationID + ", received operation ID " +
_readOperationID + ";");
117        }
118     }
119  }
```

SHTOJCA C

Kodi i serverit lidhës për JAX-WS

C.1 Kodi i serverit lidhës për Shërbime Ueb i gjeneruar nga ndërfaqja Termostat

```
 1 import javax.jws.WebService;
 2 import javax.ejb.*;
 3 import javax.jws.WebParam;
 4 import javax.xml.ws.Holder;
 5
 6 @Stateful
 7 @WebService(endpointInterface="Termostat",
targetNamespace="Termostat")
 8 public class TermostatImpl1 implements Termostat
 9 {
10
11    ...
12
13    public synchronized int get_minTempAllowed()
14    {
15
16       ...
17
18    }
19
20
21    public synchronized void setTemperature(short devID, float
temp) throws DeviceNotFound_Exception,
TermostatPackage.TempNotAllowed_Exception
22    {
23       ...
24
25       if ((_readStreamLength == _isSize) && (_readInterfaceID ==
_interfaceID) && (_readOperationID == _operationID) && (_readStatusID
== (byte) 0));
26       else if ((_readStreamLength == (byte) 6) &&
(_readInterfaceID == _interfaceID) && (_readOperationID ==
_operationID) && (_readStatusID == (byte) 1))
27          throw new DeviceNotFound_Exception("Global exception (1)
raised!", new DeviceNotFound(_inStream.readShort()));
28       else if ((_readStreamLength == (byte) 10) &&
(_readInterfaceID == _interfaceID) && (_readOperationID ==
_operationID) && (_readStatusID == (byte) 129))
29          throw new
TermostatPackage.TempNotAllowed_Exception("Local exception (129)
raised!", new TermostatPackage.TempNotAllowed(_inStream.readShort(),
_inStream.readFloat()));
```

```
30        else
31            ...
32
33    }
34 }
```

C.2 Kodi i ndërfaqes Termostat për Shërbime Ueb i gjeneruar nga skedari IDL

```
 1 import javax.jws.*;
 2 import javax.jws.soap.*;
 3 import javax.jws.soap.SOAPBinding.Style;
 4 import javax.xml.ws.Holder;
 5
 6 @WebService(targetNamespace="Termostat")
 7 @SOAPBinding(style=Style.RPC)
 8 public interface Termostat
 9 {
10    @WebMethod
11    public int get_minTempAllowed();
12    @WebMethod
13    public int get_maxTempAllowed();
14    @WebMethod
15    public void setTemperature(short devID, float temp) throws
DeviceNotFound_Exception, TermostatPackage.TempNotAllowed_Exception;
16 }
```

C.3 Kodi i gabimit lokal TempNotAllowed për Shërbime Ueb i gjeneruar nga skedari IDL

```
 1 package TermostatPackage;
 2
 3 import java.lang.Exception;
 4
 5 public class TempNotAllowed extends Exception {
 6    public short deviceID;
 7    public float temperature;
 8
 9    public TempNotAllowed(short deviceID, float temperature) {
10       this.deviceID = deviceID;
11       this.temperature = temperature;
12    }
13 }
```

C.4 Kodi i klasës fault TempNotAllowed_Exception për gabimin lokal TempNotAllowed, për Shërbime Ueb i gjeneruar nga skedari IDL

```
 1 package TermostatPackage;
 2
 3 import javax.xml.ws.WebFault;
 4
 5 @WebFault(name = "TempNotAllowed", targetNamespace = "Termostat")
 6 public class TempNotAllowed_Exception extends Exception {
 7    private TempNotAllowed faultInfo;
 8
 9    public TempNotAllowed_Exception(String message, TempNotAllowed
faultInfo) {
10       super(message);
11       this.faultInfo = faultInfo;
12    }
13
14    public TempNotAllowed_Exception(String message, TempNotAllowed
faultInfo, Throwable cause) {
15       super(message, cause);
16       this.faultInfo = faultInfo;
17    }
18
19    public TempNotAllowed getFaultInfo() {
20       return faultInfo;
21    }
22 }
```

SHTOJCA D

Kodi i klasave ndihmëse MISPSerialManaged dhe StreamOperations2

D.1 Metoda connect e klasës MISPSerialManaged

```
 1    public boolean connect()
 2    {
 3       boolean success = true;
 4
 5       int _noOfRetries = 0;
 6       while (_noOfRetries < maxNumOfConnRetries)
 7          try
 8          {
 9             if (_noOfRetries > 0)
10                System.out.println("Retrying to connect at " +
this.comPort + " port ...");
11             success = this.openPort();
12             break;
13          }
14          catch(Exception _xp)
15          {
16             _noOfRetries++;
17             System.out.println(_xp);
18             try
19             {
20                Thread.sleep(maxTimeBetwConnRetries);
21             }
22             catch(Exception _xpsleep)
23             {
24                System.out.println(_xpsleep);
25             }
26          }
27       if (_noOfRetries >= maxNumOfConnRetries) success = false;
28
29       isConnected = success;
30       return success;
31    }
```

D.2 Sekuencë nga metoda serialEvent e klasës MISPSerialManaged

```
1 case SerialPortEvent.DATA_AVAILABLE:
2 // we get here if data has been received
```

```
 3     byte[] readBuffer = new byte[BUFSIZE];
 4     actualBytesRead = 0;
 5     try {
 6        // read data
 7        while (inputStream.available() > 0) {
 8           actualBytesRead=inputStream.read(readBuffer);
 9
10           if(streamLength==0)
11              streamLength=readBuffer[0];
12
13           if(totalBytesRead+actualBytesRead>streamLength)
14           {
15              for(int place_indx=0;place_indx<streamLength-
totalBytesRead; place_indx++)
16
readStream[place_indx+totalBytesRead]=readBuffer[place_indx];
17
18              isDataReady = true;
19              // Read the new stream for partial reading
20              for(int place_indx=streamLength-
totalBytesRead;place_indx<actualBytesRead; place_indx++)
21                 readStream[place_indx-(streamLength-
totalBytesRead)]=readBuffer[place_indx];
22              // Set new stream parameters
23              streamLength = readBuffer[streamLength-
totalBytesRead];
24              totalBytesRead = actualBytesRead-(streamLength-
totalBytesRead);
25           }
26           else if(totalBytesRead+actualBytesRead==streamLength)
27           {
28              for(int place_indx=0;place_indx<actualBytesRead;
place_indx++)
29
readStream[place_indx+totalBytesRead]=readBuffer[place_indx];
30              totalBytesRead=totalBytesRead+actualBytesRead;
31
32              isDataReady = true;
33              // Reset and prepare for new stream
34              streamLength = 0;
35              totalBytesRead = 0;
36           }
37           else
38           {
39              for(int place_indx=0;place_indx<actualBytesRead;
place_indx++)
40
readStream[place_indx+totalBytesRead]=readBuffer[place_indx];
41              totalBytesRead=totalBytesRead+actualBytesRead;
42           }
43        }
44     }
```

D.3 Metoda disconnect e klasës MISPSerialManaged

```
1     public boolean disconnect()
2     {
3        boolean success = true;
4        if (this.closePort())
5           isConnected = false;
6        return success;
7     }
```

D.4 Konstruktorët e klasës StreamOperations2

```
 1     public StreamOperations2(byte streamSize)
 2     {
 3        stream = new byte[streamSize];
 4        stream[0] = streamSize;
 5        indxPtr = 1;
 6     }
 7
 8     public StreamOperations2(byte [] inStream)
 9     {
10        stream = inStream;
11        indxPtr = 0;
12     }
```

D.5 Operacionet e shkrimit (writeInt) dhe leximit (readInt) në stream të tipit int të klasës StreamOperations2

```
 1     public void writeInt(int iVal)
 2     {
 3        stream[indxPtr++] = (byte)((iVal >> 24) & 0xff);
 4        stream[indxPtr++] = (byte)((iVal >> 16) & 0xff);
 5        stream[indxPtr++] = (byte)((iVal >> 8) & 0xff);
 6        stream[indxPtr++] = (byte)(iVal & 0xff);
 7     }
 8
 9     public int readInt()
10     {
11        int retVal = ((stream[indxPtr] & 0xff) << 24) |
((stream[indxPtr+1] & 0xff) << 16) | ((stream[indxPtr+2] & 0xff) <<
8) | (stream[indxPtr+3] & 0xff);
12        indxPtr += 4;
13        return retVal;
14     }
```

D.6 Operacionet e shkrimit (writeBoolean) dhe leximit (readBoolean) në stream të tipit boolean të klasës StreamOperations2

```
1     public void writeBoolean(boolean blVal)
2     {
3        if (blVal)
4           stream[indxPtr] = (byte) 1;
5        else
6           stream[indxPtr] = (byte) 0;
7
8       indxPtr++;
9     }
10
11    public boolean readBoolean()
12    {
13       boolean retVal = false;
14       if (stream[indxPtr++] != 0) retVal = true;
15       return retVal;
16    }
```

D.7 Operacionet e shkrimit (writeFloat) dhe leximit (readFloat) në stream të tipit float të klasës StreamOperations2

```
1     public void writeFloat(float fVal)
2     {
3        int bits = Float.floatToIntBits(fVal);
4        stream[indxPtr++] = (byte)((bits >> 24) & 0xff);
5        stream[indxPtr++] = (byte)((bits >> 16) & 0xff);
6        stream[indxPtr++] = (byte)((bits >> 8) & 0xff);
7        stream[indxPtr++] = (byte)(bits & 0xff);
8     }
9
10    public float readFloat()
11    {
12       byte[] b = new byte[]{stream[indxPtr], stream[indxPtr+1],
stream[indxPtr+2], stream[indxPtr+3]};
13       indxPtr += 4;
14       return  ByteBuffer.wrap(b).getFloat();
15    }
```

SHTOJCA E

Kodi i gramatikave, aksioneve semantike, dhe klasave të nevojshme për ndërtim të listës së objekteve

E.1 Gramatika përkufizuese e skedarit implementues

```
 1 IMPL8051 = { "implement" identifier (.
 2    if (idle == null)
 3    {
 4       idle = new IDLElement();
 5       firstidle = idle;
 6    }
 7    else
 8    {
 9       IDLElement previdle = idle;
10       idle = new IDLElement();
11       previdle.next = idle;
12       idle.previous = previdle;
13    }
14
15         idle.type = 1;
16         idle_intrfc = IDLEntities.getInterface(t.val,
firstidlefromidl);
17         idle.object = idle_intrfc;
18     .) { "," identifier (.
19    IDLElement previdle = idle;
20    idle = new IDLElement();
21    previdle.next = idle;
22    idle.previous = previdle;
23
24    idle.type = 1;
25    idle_intrfc = IDLEntities.getInterface(t.val,
firstidlefromidl);
26    idle.object = idle_intrfc;
27    .) } ";" }.
```

E.2 Pjesë nga gramatika e skedarit IDL me aksione semantike – regjistrimi i ndërfaqes në listë të objekteve

```
1 interfacedef (.
2    if (idle == null)
```

```
    {
       idle = new IDLElement();
       firstidle = idle;
    }
    else
    {
       IDLElement oldidle = idle;
       idle = new IDLElement();
       oldidle.next = idle;
       idle.previous = oldidle;
    }

    idle.type = 1;
    idle_intrfc = new IDLEInterface();
    idle.object = idle_intrfc;

    lclExcptCnt = (short)129;
    .)
```

E.3 Pjesë nga gramatika e skedarit IDL me aksione semantike – regjistrimi i operacionit në kuadër të ndërfaqes në listë të objekteve

```
standardmethod (.
    if (idle_intrfc.currentie == null)
    {
       idle_intrfc.currentie = new InterfaceElement();
       idle_intrfc.firstie = idle_intrfc.currentie;
    }
    else
    {
       InterfaceElement oldie = idle_intrfc.currentie;
       idle_intrfc.currentie = new InterfaceElement();
       oldie.next = idle_intrfc.currentie;
       idle_intrfc.currentie.previous = oldie;
    }

    idle_intrfc.currentie.type = 1;
    st_oper = new IEStandardOperation();
    idle_intrfc.currentie.object = st_oper;

.) = simpletypesplusvoid <out short tid>  (.
    st_oper.retParam = new Parameter();
    st_oper.retParam.direction = 0;
    st_oper.retParam.type = tid;
    st_oper.retParam.size =
IDLEntities.getBasicDataTypeSize(st_oper.retParam.type);
.) identifier (.
    st_oper.operationID = t.val;
.) "(" [methodparammetersbody] ")" [ "raises" "(" identifier (.
    st_oper.excptn = new IDLEExceptionList();
    st_oper.firstExcptn = st_oper.excptn;
```

```
29    st_oper.excptn.exception = IDLEntities.getException(t.val,
firstidle, idle_intrfc);
30 .) { "," identifier (.
31    IDLEExceptionList prevxps = st_oper.excptn;
32    st_oper.excptn = new IDLEExceptionList();
33    prevxps.next = st_oper.excptn;
34    st_oper.excptn.previous = prevxps;
35    st_oper.excptn.exception = IDLEntities.getException(t.val,
firstidle, idle_intrfc);
36 .) } ")" ] ";".
```

E.4 Pjesë nga gramatika e skedarit IDL me aksione semantike – regjistrimi i parametrave (argumenteve) të operacionit standard

```
 1 methodparammetersbody = methodparammeter { "," methodparammeter}.
 2 methodparammeter (.
 3    if (st_oper.params == null)
 4    {
 5       st_oper.params = new Parameter();
 6       st_oper.firstParam = st_oper.params;
 7    }
 8    else
 9    {
10       Parameter old_prm = st_oper.params;
11       st_oper.params = new Parameter();
12       old_prm.next = st_oper.params;
13       st_oper.params.previous = old_prm;
14    }
15
16 .) = ("in" (. st_oper.params.direction = 1; .)
17 | "out" (. st_oper.params.direction = 2; .)
18 | "inout" (. st_oper.params.direction = 3; .))
19 simpletypes <out short tpid>
20 (.
21    st_oper.params.type = tpid;
22    st_oper.params.size =
IDLEntities.getBasicDataTypeSize(st_oper.params.type);
23 .) identifier (. st_oper.params.name = t.val; .).
```

E.5 Funksioni getInterface i klasës IDLEntities për kthim të instancës së klasës së ndërfaqes nga lista e objekteve në bazë të emrit të tij

```
 1 public static IDLEInterface getInterface(String intfcName,
IDLElement idle)
 2 {
```

```
 3     IDLEInterface retVal = null;
 4
 5     IDLElement idlecpy = idle;
 6     while(idlecpy != null)
 7     {
 8        if(idlecpy.type == 1)
 9        {
10           IDLEInterface locIntrfc = (IDLEInterface)idlecpy.object;
11           if (locIntrfc.interfaceName.equals(intfcName)) retVal =
locIntrfc;
12        }
13
14        idlecpy = idlecpy.next;
15     }
16
17     return retVal;
18  }
```

E.6 Funksioni getException i klasës IDLEntities për kthim të instancës së klasës së gabimit nga lista e objekteve dhe ndërfaqja e dhënë në bazë të emrit të tij

```
 1 public static IDLEException getException(String excptnName,
IDLElement idle, IDLEInterface idlei)
 2 {
 3     IDLEException retVal = null;
 4
 5     IDLElement idlecpy = idle;
 6     while(idlecpy != null)
 7     {
 8        if(idlecpy.type == 2)
 9        {
10           IDLEException locExc = (IDLEException)idlecpy.object;
11           if (locExc.excptName.equals(excptnName)) retVal = locExc;
12        }
13
14        idlecpy = idlecpy.next;
15     }
16
17     InterfaceElement iecpy = idlei.firstie;
18     while(iecpy != null)
19     {
20        if(iecpy.type == 4)
21        {
22           IDLEException locExc = (IDLEException)iecpy.object;
23           if (locExc.excptName.equals(excptnName)) retVal = locExc;
24        }
25
26        iecpy = iecpy.next;
27     }
```

```
28
29      return retVal;
30  }
```

E.7 Klasa e listës globale të objekteve IDLElement

```
1  package CORBAServerCodeGenerator2;
2
3  public class IDLElement{
4     public int type;
5     public Object object = null;
6     public IDLElement first = null, previous = null, next = null;
7  }
```

E.8 Klasa e ndërfaqes IDLEInterface

```
1  package CORBAServerCodeGenerator2;
2
3  public class IDLEInterface{
4     public String interfaceName = null;
5     public boolean isSEASecured = false;
6     public boolean generateOperationID = false;
7     public InterfaceElement firstie = null, currentie = null;
8  }
```

E.9 Klasa e elementeve të ndërfaqes InterfaceElement

```
1  package CORBAServerCodeGenerator2;
2
3  public class InterfaceElement{
4     public InterfaceElement previous = null;
5     public int type;
6     public Object object = null;
7     public InterfaceElement next = null;
8  }
```

E.10 Klasa e operacionit standard IEStandardOperation

```
1  package CORBAServerCodeGenerator2;
```

```
 2
 3 public class IEStandardOperation{
 4    public Parameter retParam = null;
 5    public String operationID = null;
 6    public Parameter firstParam = null;
 7    public Parameter params = null;
 8    public IDLEExceptionList firstExcptn = null, excptn = null;
 9    public boolean generateStatusID = false;
10 }
```

E.11 Klasa e parametrit Parameter

```
 1 package CORBAServerCodeGenerator2;
 2
 3 public class Parameter{
 4    public Parameter previous = null;
 5    public byte direction;
 6    public short type;
 7    public short size;
 8    public String name;
 9    public Parameter next = null;
10 }
```

E.12 Klasa e listës së gabimeve IDLEExceptionList

```
1 package CORBAServerCodeGenerator2;
2
3 public class IDLEExceptionList{
4    public IDLEExceptionList previous = null;
5    public IDLEException exception = null;
6    public IDLEExceptionList next = null;
7 }
```

E.13 Klasa e gabimit IDLEException

```
1 package CORBAServerCodeGenerator2;
2
3 public class IDLEException{
4    public short excptID = 0;
5    public String excptName = null;
6    public Parameter firstParam = null, params = null;
7 }
```

E.14 Klasa e operacionit njëkahor IEOnewayOperation

```
1 package CORBAServerCodeGenerator2;
2
3 public class IEOnewayOperation{
4    public String operationID = null;
5    public Parameter firstParam = null;
6    public Parameter params = null;
7 }
```

E.15 Klasa e atributit IEAttribute

```
1 package CORBAServerCodeGenerator2;
2
3 public class IEAttribute{
4    public boolean isReadOnly = false;
5    public Parameter attribType = null;
6    public String attributeID = null;
7 }
```

SHTOJCA F

Kodi i modifikimit të parserit

F.1 Metoda simpletypesplusvoid brenda klasës Parser të gramatikës së skedarit IDL

```
1    short  simpletypesplusvoid() {
2       short  tid = 0;
3       if (StartOf(1)) {
4          short tpid = tid = simpletypes();
5       } else if (la.kind == 11) {
6          tid = 1;
7          Get();
8       } else SynErr(33);
9       return tid;
10   }
```

F.2 Metoda simpletypes brenda klasës Parser të gramatikës së skedarit IDL

```
1    short  simpletypes() {
2       short  tpid = 0;
3       switch (la.kind) {
4       case 16: {
5          tpid = 2;
6          Get();
7          break;
8       }
9       case 17: {
10         tpid = 3;
11         Get();
12         break;
13      }
14      case 18: {
15         tpid = 4;
16         Get();
17         break;
18      }
19      case 19: {
20         tpid = 5;
21         Get();
22         break;
23      }
24      case 20: {
25         tpid = 9;
```

```
26          Get();
27          break;
28       }
29       case 21: {
30          tpid = 12;
31          Get();
32          break;
33       }
34       case 22: {
35          tpid = 7;
36          Get();
37          if (la.kind == 21 || la.kind == 22) {
38             if (la.kind == 22) {
39                tpid = 10;
40                Get();
41             } else {
42                tpid = 13;
43                Get();
44             }
45          }
46          break;
47       }
48       case 23: {
49          Get();
50          if (la.kind == 19) {
51             tpid = 6;
52             Get();
53          } else if (la.kind == 22) {
54             tpid = 8;
55             Get();
56             if (la.kind == 22) {
57                tpid = 11;
58                Get();
59             }
60          } else SynErr(34);
61          break;
62       }
63       default: SynErr(35); break;
64       }
65       return tpid;
66    }
```

SHTOJCA G

Kodi i klasës kryesore ku thirren gjeneratorët

G.1 Sekuencë nga metoda decideWhatToGenerate ku vendoset nëse duhet gjeneruar fusha e numrit identifikues të operacionit

```
 1 int noOfOperationsDeclared = 0;
 2 InterfaceElement ie_call = i.firstie;
 3 while (ie_call != null)
 4 {
 5    switch(ie_call.type)
 6    {
 7       case 1:
 8             IEStandardOperation stop = (IEStandardOperation)
ie_call.object;
 9             stop.generateStatusID = (stop.firstExcptn != null);
10             noOfOperationsDeclared++;
11             break;
12       case 2:
13             noOfOperationsDeclared++;
14             break;
15       case 3:
16             IEAttribute attr = (IEAttribute) ie_call.object;
17             if (!attr.isReadOnly)
18                noOfOperationsDeclared += 2;
19             else
20                noOfOperationsDeclared++;
21             break;
22    }
23
24    ie_call = ie_call.next;
25 }
26 i.generateOperationID = (noOfOperationsDeclared > 1) ||
forceOperationIDGeneration;
```

SHTOJCA H

Kodi i gjeneratorit të serverit 8051

H.1 Klasa e gjeneratorit të serverit në mikrokontrollor 8051

CodeGenerator8051Server2

```
 1 package CORBAServerCodeGenerator2;
 2
 3 import java.io.*;
 4 import java.util.*;
 5
 6 public class CodeGenerator8051Server2
 7 {
 8    public CodeGenerator8051Server2(String fileName, IDLElement
firstIDLE, boolean generateInterfaceID, long clockRate, int baudRate,
byte initialRAMOffset, boolean genCompAttr);
 9
10    private boolean preCalculations();
11
12    private short max(short val1, short val2);
13
14    public void generate();
15
16    private String processInterface(IDLEInterface intrfc,
IDLEInterface nextIntrfc);
17
18    private String processStandardOperation(IDLEInterface i, int
interfaceID, IEStandardOperation operation, InterfaceElement
nextEntity);
19
20    private void processOnewayOperation(IDLEInterface i, int
interfaceID, IEOnewayOperation operation, InterfaceElement
nextEntity);
21
22    private String processAttribute(IDLEInterface i, int
interfaceID, IEAttribute attr, InterfaceElement nextEntity);
23
24    private String getNextOpID(IDLEInterface i, int interfaceID,
InterfaceElement element);
25
26    private void writeFreeRAMInfo();
27
28    private String getNextInterfaceCheckID(IDLElement nextIE, int
currentInterfaceID);
29 }
```

H.2 Sekuencë nga metoda preCalculations ku bëhet llogaritja e gjatësisë maksimale të mesazheve të operacionit standard në ndërfaqen e dhënë

```
 1 IEStandardOperation stop = (IEStandardOperation) ie_call.object;
 2 short stopInMsgLen = (short)(1 + (generateInterfaceID ? 1: 0) +
(i.generateOperationID ? 1: 0));
 3 short stopOutMsgLen = 0;
 4 if (stop.retParam != null) stopOutMsgLen += stop.retParam.size;
 5 Parameter stopPrm = stop.firstParam;
 6 while(stopPrm != null)
 7 {
 8    if ((stopPrm.direction & 1) > 0)
 9       stopInMsgLen += stopPrm.size;
10    if ((stopPrm.direction & 2) > 0)
11       stopOutMsgLen += stopPrm.size;
12    stopPrm = stopPrm.next;
13 }
14 maxMsgLen = max(maxMsgLen, stopInMsgLen);
15 maxMsgLen = max(maxMsgLen, stopOutMsgLen);
16 maxEntID++;
17 entitiesCount++;
18 // Calculate exception message size(s)
19 if (stop.firstExcptn != null)
20 {
21    IDLEExceptionList stopXps = stop.firstExcptn;
22    while(stopXps != null)
23    {
24       Parameter xpsPrm = stopXps.exception.firstParam;
25       stopOutMsgLen = 0;
26       while(xpsPrm != null)
27       {
28          stopOutMsgLen += xpsPrm.size;
29          xpsPrm = xpsPrm.next;
30       }
31       maxMsgLen = max(maxMsgLen, stopOutMsgLen);
32       stopXps = stopXps.next;
33    }
34 }
```

H.3 Sekuencë nga metoda preCalculations ku bëhet llogaritja e gjatësisë maksimale të mesazheve të atributit në ndërfaqen e dhënë

```
 1 IEAttribute attr = (IEAttribute) ie_call.object;
 2 short attrGetInMsgLen = (short)(1 + (generateInterfaceID ? 1: 0) +
(i.generateOperationID ? 1: 0));
 3 short attrGetOutMsgLen = attr.attribType.size;
```

```
 4 maxMsgLen = max(maxMsgLen, attrGetInMsgLen);
 5 maxMsgLen = max(maxMsgLen, attrGetOutMsgLen);
 6 if (!attr.isReadOnly)
 7 {
 8    short attrSetInMsgLen = (short)((short)(1 +
(generateInterfaceID ? 1: 0) + (i.generateOperationID ? 1: 0)) +
attr.attribType.size);
 9    maxMsgLen = max(maxMsgLen, attrSetInMsgLen);
10    maxEntID++;
11    entitiesCount++;
12 }
13 attrOffset += attr.attribType.size;
14 maxEntID++;
15 entitiesCount++;
```

H.4 Sekuencë nga metoda generate e klasës CodeGenerator8051Server2

```
 1 ...
 2 pw.println("ORG 0030H");
 3 pw.println("InitSerComAndEnblSerInt:");
 4 pw.println("MOV SCON, #50H");
 5 pw.println("MOV TMOD, #20H");
 6 pw.println("MOV TH1, #" + this.TH1 + " ; Set " + this.baudRate + "
baud rate");
 7 pw.println("SETB TR1 ; Start timer 1");
 8 ...
 9 if (this.generateInterfaceID)
10 {
11    pw.println("; Read interface ID from RAM location " +
(this.startRAMLocation + 1));
12    pw.println("MOV R0, #" + (this.startRAMLocation + 1));
13    pw.println("MOV A, @R0");
14 }
15
16 IDLElement ie_call = this.firstIDLE;
17 while(ie_call != null)
18 {
19    if (ie_call.type == 1)
20    {
21       IDLEInterface nextIntrfc = null;
22       if (ie_call.next != null)
23          if (ie_call.next.type == 1)
24             nextIntrfc = (IDLEInterface)ie_call.next.object;
25
26       DB_contents +=
processInterface((IDLEInterface)ie_call.object, nextIntrfc);
27    }
28
29    ie_call = ie_call.next;
30 }
31 ...
32 pw.println("SendResponseBodyRAM:");
33 pw.println("MOV R0, #" + this.startRAMLocation);
```

```
34 pw.println("SendResponseBodyByte:");
35 pw.println("MOV A, @R0");
36 pw.println("MOV SBUF, A");
37 pw.println("JNB TI, $");
38 pw.println("CLR TI");
39 pw.println("INC R0");
40 pw.println("DJNZ R1, SendResponseBodyByte");
41 pw.println("RET");
42 if (DB_contents.trim().length() > 0)
43    pw.println(DB_contents);
44 pw.println("");
45 pw.print("END");
46
47 pw.close();
```

H.5 Metoda processInterface e klasës CodeGenerator8051Server2

```
 1 private String processInterface(IDLEInterface intrfc,
IDLEInterface nextIntrfc)
 2 {
 3    String retVal = "";
 4    this.currentOperationID = 0;
 5
 6    if (this.generateInterfaceID)
 7    {
 8       pw.println("");
 9       pw.println("ChckIntID_" + intrfc.interfaceName + "_" +
this.currentInterfaceID + ":");
10       pw.println("CJNE A, #" + this.currentInterfaceID + ",
ChckIntID_" + intrfc.interfaceName + "_" + this.currentInterfaceID +
"_Failed ; Check interface ID");
11
12       if (intrfc.generateOperationID)
13          pw.println("JMP Chck_" + intrfc.interfaceName + "_" +
this.currentInterfaceID + "_operationIDs");
14       else
15          pw.println("JMP " + getNextOpID(intrfc,
this.currentInterfaceID, intrfc.firstie));
16
17       pw.println("ChckIntID_" + intrfc.interfaceName + "_" +
this.currentInterfaceID + "_Failed:");
18       if (nextIntrfc == null)
19          pw.println("JMP EnSerComAndInt");
20       else
21          pw.println("JMP " + "ChckIntID_" +
nextIntrfc.interfaceName + "_" + (this.currentInterfaceID + 1));
22    }
23
24    if (intrfc.generateOperationID)
25    {
26       pw.println("");
27       pw.println("; Read operation ID from RAM location " +
(this.startRAMLocation + (generateInterfaceID ? 1: 0) + 1));
```

```
28        pw.println("Chck_" + intrfc.interfaceName + "_" +
this.currentInterfaceID + "_operationIDs:");
29        pw.println("MOV R0, #" + (this.startRAMLocation +
(generateInterfaceID ? 1: 0) + 1));
30        pw.println("MOV A, @R0");
31     }
32
33     InterfaceElement ie_call = intrfc.firstie;
34
35     while(ie_call != null)
36     {
37        switch(ie_call.type)
38        {
39           case 1:
40              retVal += processStandardOperation(intrfc,
currentInterfaceID, (IEStandardOperation) ie_call.object,
ie_call.next);
41              break;
42           case 2:
43              processOnewayOperation(intrfc, currentInterfaceID,
(IEOnewayOperation) ie_call.object, ie_call.next);
44              break;
45           case 3:
46              retVal += processAttribute(intrfc, currentInterfaceID,
(IEAttribute) ie_call.object, ie_call.next);
47              break;
48        }
49
50        ie_call = ie_call.next;
51     }
52
53     this.currentInterfaceID++;
54
55     return retVal;
56  }
```

H.6 Sekuencë nga metoda processStandardOperation e klasës CodeGenerator8051Server2 ku gjenerohet prova e adresimit të operacionit standard

```
 1  if (i.generateOperationID)
 2  {
 3     pw.println("Chk_IntImpl_" + i.interfaceName + "_" + interfaceID
+ "_STOP_" + operation.operationID + "_ID:");
 4     pw.println("CJNE A, #" + this.currentOperationID + ",
Chk_IntImpl_" + i.interfaceName + "_" + interfaceID + "_STOP_" +
operation.operationID + "_ID_Failed ; Check operation ID");
 5     pw.println("JMP IntImpl_" + i.interfaceName + "_" + interfaceID
+ "_STOP_" + operation.operationID + "_ID");
```

```
 6     pw.println("Chk_IntImpl_" + i.interfaceName + "_" + interfaceID
+ "_STOP_" + operation.operationID + "_ID_Failed:");
 7     String nextOpID = getNextOpID(i, interfaceID, nextEntity);
 8     if (nextOpID.equals("EnSerComAndInt"))
 9        pw.println("JMP EnSerComAndInt");
10     else
11        pw.println("JMP Chk_" + nextOpID);
12  }
```

H.7 Sekuencë nga metoda processStandardOperation e klasës CodeGenerator8051Server2 ku gjenerohen adresat e pritura të parametrave hyrës dhe hyrës/dalës

```
 1 boolean inTextIsNotWriten = true;
 2 short actualInPrmRAMLoc = (short)(this.startRAMLocation + 1 +
(generateInterfaceID ? 1: 0) + (i.generateOperationID ? 1: 0));
 3 Parameter stopInPrm = operation.firstParam;
 4 while(stopInPrm != null)
 5 {
 6    if ((stopInPrm.direction & 1) > 0)
 7    {
 8       if (inTextIsNotWriten)
 9       {
10          pw.println("; Please find in and inout parameters at RAM
locations as follows:");
11          inTextIsNotWriten = false;
12       }
13    pw.print("; parameter, name: " + stopInPrm.name + ", direction:
" + IDLEntities.getParamterDirectionNameFromID(stopInPrm.direction) +
", CORBA type: " + IDLEntities.getBasicDataTypeName(stopInPrm.type) +
", size: " + stopInPrm.size + ",");
14    if (stopInPrm.size == 1)
15       pw.println(" RAM location: " + actualInPrmRAMLoc);
16    else
17       pw.println(" RAM locations: " + actualInPrmRAMLoc + ".." +
(actualInPrmRAMLoc + stopInPrm.size - 1));
18    actualInPrmRAMLoc += stopInPrm.size;
19    }
20
21    stopInPrm = stopInPrm.next;
22 }
```

H.8 Sekuencë nga metoda processStandardOperation e klasës CodeGenerator8051Server2 ku gjenerohen komentet si të sinjalizohen gabimet

```
 1 if (operation.firstExcptn != null)
 2 {
 3    pw.println("; Uncomment the following code to terminate
sucessfully at the end of your code");
 4    pw.println("; MOV A, #0");
 5    pw.println("; JMP DcdRspns" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") +
(i.generateOperationID ? "_STOP_" + operation.operationID : ""));
 6    IDLEExceptionList exc_lst = operation.firstExcptn;
 7    while (exc_lst != null)
 8    {
 9       IDLEException exc = exc_lst.exception;
10       pw.println("; In case the operation terminates with " +
exc.excptName  + " exception ...");
11       if (exc.firstParam != null)
12       {
13          pw.println("; Please place param(s) at following RAM
addresses:");
14          Parameter excprm  = exc.firstParam;
15          short actualExcPrmRAMLoc = this.startRAMLocation;
16          while (excprm != null)
17          {
18             pw.print("; parameter, name: " + excprm.name + ",
size: " + excprm.size + ",");
19             if (excprm.size == 1)
20                pw.println(" RAM location: " + actualExcPrmRAMLoc);
21             else
22                pw.println(" RAM locations: " + actualExcPrmRAMLoc
+ ".." + (actualExcPrmRAMLoc + excprm.size - 1));
23             actualExcPrmRAMLoc += excprm.size;
24             excprm = excprm.next;
25          }
26       }
27       pw.println("; Uncomment the following code to terminate with
exception " + exc.excptName  + " (ID: " + exc.excptID  + ") at the
end of your code");
28       pw.println("; MOV A, #" + exc.excptID);
29       pw.println("; JMP DcdRspns" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") +
(i.generateOperationID ? "_STOP_" + operation.operationID : ""));
30
31       exc_lst = exc_lst.next;
32    }
33 }
```

H.9 Sekuencë nga metoda processStandardOperation e klasës CodeGenerator8051Server2 ku gjenerohen instruksionet që provojnë nëse është sinjalizuar gabimi i caktuar dhe ndërtohet stream-i i secilit gabim

```
 1 IDLEExceptionList exc_lst = operation.firstExcptn;
 2 while (exc_lst != null)
 3 {
 4    IDLEException exc = exc_lst.exception;
 5 
 6 
 7    short totalExcPrmSize = 0;
 8    if (exc.firstParam != null)
 9    {
10       Parameter excprm  = exc.firstParam;
11 
12       while (excprm != null)
13       {
14          totalExcPrmSize += excprm.size;
15          excprm = excprm.next;
16       }
17    }
18 
19    pw.println("DcdRspns_" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc.excptID + ":");
20    pw.println("CJNE A, #" + exc.excptID + ", DcdRspns_" +
(this.generateInterfaceID ? "_IntImpl_" + i.interfaceName + "_" +
interfaceID: "") + "_STOP_" + operation.operationID + "_STTS_" +
exc.excptID + "_Failed");
21    pw.println("JMP DcdRspns_" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc.excptID + "_RSPNS");
22    pw.println("DcdRspns_" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc.excptID + "_Failed:");
23    if (exc_lst.next != null)
24       pw.println("JMP DcdRspns_" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc_lst.next.exception.excptID);
25    else
26       pw.println("JMP EnSerComAndInt ; Jump to restore point");
27    pw.println("DcdRspns_" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc.excptID + "_RSPNS:");
28    pw.println("; Send response header for status exception (" +
exc.excptID + ")");
29    pw.println("MOV R0, #" + (1 + (this.generateInterfaceID ? 1 :
0) + (i.generateOperationID ? 1 : 0) + 1));
30    pw.println("MOV DPTR, #DB" + (this.generateInterfaceID ?
"_IntImpl_" + i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc.excptID + "_resp_head");
```

```
31    retVal += "\r\nDB" + (this.generateInterfaceID ? "_IntImpl_" +
i.interfaceName + "_" + interfaceID: "") + "_STOP_" +
operation.operationID + "_STTS_" + exc.excptID + "_resp_head: DB " +
(1 + (this.generateInterfaceID ? 1 : 0) + (i.generateOperationID ? 1
: 0) + 1 + totalExcPrmSize) + (this.generateInterfaceID ? ", " +
interfaceID : "") + (i.generateOperationID ? ", " +
this.currentOperationID : "") + ", " + exc.excptID;
32    pw.println("CALL SendResponseHeader");
33    if (totalExcPrmSize > 0)
34    {
35       pw.println("; Send response body (return, out and inout
parameters) from RAM");
36       pw.println("MOV R1, #" + totalExcPrmSize);
37       pw.println("CALL SendResponseBodyRAM");
38    }
39    pw.println("JMP EnSerComAndInt ; Jump to restore point");
40
41    exc_lst = exc_lst.next;
42 }
```

SHTOJCA I

Kodi i gjeneratorit të serverit lidhës për JacORB

I.1 Klasa e gjeneratorit të serverit lidhës për JacORB

CodeGeneratorJacORBBindServer2

```
 1 package CORBAServerCodeGenerator2;
 2
 3 import java.io.*;
 4 import java.util.*;
 5
 6 public class CodeGeneratorJacORBBindServer2
 7 {
 8    public CodeGeneratorJacORBBindServer2(IDLElement firstIDLE,
boolean generateInterfaceID, String filePath);
 9
10    public void generate() throws IOException;
11
12    private void writeConnect(boolean hasReturnVal);
13
14    private void writeDisconnect();
15
16    private void processStandardOperation(IEStandardOperation
operation, IDLEInterface intrfc);
17
18    private void processOnewayOperation(IEOnewayOperation
operation, IDLEInterface intrfc);
19
20    private void processAttribute(IEAttribute attr, IDLEInterface
intrfc);
21 }
```

I.2 Sekuencë nga metoda generate e klasës

CodeGeneratorJacORBBindServer2

```
 1 IDLElement ie_call = this.firstIDLE;
 2 while(ie_call != null)
 3 {
 4    if (ie_call.type == 1)
 5    {
 6       IDLEInterface intrfc = (IDLEInterface)ie_call.object;
```

```
 7       String interfaceIDtoString = this.generateInterfaceID ?
this.interfaceID + "" : "";
 8       this.pw = new PrintWriter(new FileWriter(this.filePath +
"\\" + intrfc.interfaceName + "Impl" + interfaceIDtoString +
".java"));
 9       this.currentOperationID = (short) 0;
10
11       ...
12       pw.println("class " + intrfc.interfaceName + "Impl" +
interfaceIDtoString + " extends " + intrfc.interfaceName + "POA");
13       pw.println("{");
14       pw.println("\tMISPSerialManaged _ms;");
15       if (this.generateInterfaceID)
16          pw.println("\tbyte _interfaceID = (byte) " +
this.interfaceID + ";");
17       pw.println("\tConnectionPolicy _connPolicy =
ConnectionPolicy." + (this.generateInterfaceID ? "CONNECTONDEMAND" :
"ALWAYSCONNECTED") + ";");
18       pw.println("");
19       ...
20       pw.println("\tpublic " + intrfc.interfaceName + "Impl" +
interfaceIDtoString + "(String comPort, int baudRate,
ConnectionPolicy connPolicy)");
21       pw.println("\t{");
22       pw.println("\t\ttry");
23       pw.println("\t\t{");
24       pw.println("\t\t\t_ms = new MISPSerialManaged(comPort,
baudRate);");
25       pw.println("\t\t\tthis._connPolicy = connPolicy;");
26       pw.println("\t\t}");
27       pw.println("\t\tcatch(Exception _xp)");
28       pw.println("\t\t{");
29       pw.println("\t\t\tSystem.out.println(_xp);");
30       pw.println("\t\t\tSystem.exit(0);");
31       pw.println("\t\t}");
32       pw.println("\t}");
33
34       InterfaceElement iel_call = (InterfaceElement)
intrfc.firstie;
35       while(iel_call != null)
36       {
37          if ((iel_call.type >= 1) && (iel_call.type <= 3))
38             pw.println("");
39
40          switch(iel_call.type)
41          {
42             case 1:
43                processStandardOperation((IEStandardOperation)
iel_call.object, intrfc);
44                break;
45             case 2:
46                processOnewayOperation((IEOnewayOperation)
iel_call.object, intrfc);
47                break;
48             case 3:
49                processAttribute((IEAttribute) iel_call.object,
intrfc);
50                break;
```

```
51          }
52
53          iel_call = iel_call.next;
54       }
55
56       pw.print("}");
57       pw.close();
58
59       this.interfaceID++;
60    }
61
62    ie_call = ie_call.next;
63 }
```

I.3 Sekuencë nga fillimi i metodës processStandardOperation e klasës CodeGeneratorJacORBBindServer2

```
 1 private void processStandardOperation(IEStandardOperation
operation, IDLEInterface intrfc)
 2 {
 3    Parameter stopPrm = operation.firstParam;
 4    String operStart = "";
 5    String operBodyStart = "";
 6    String prmArgs = "";
 7    int prmCount = 0;
 8    String osOperations = "";
 9    String isOperations = "";
10    final int osHeaderSize = 1 + (this.generateInterfaceID ? 1 : 0)
+ (intrfc.generateOperationID ? 1 : 0);
11    final int isHeaderSize = 1 + (this.generateInterfaceID ? 1 : 0)
+ (intrfc.generateOperationID ? 1 : 0) + (operation.generateStatusID
? 1 : 0);
12    int osSize = osHeaderSize;
13    int isSize = isHeaderSize;
14    boolean hasReturnVal = false;
15
16    if (operation.retParam.type > 1)
17    {
18       isSize += operation.retParam.size;
19       operStart = "\t" + "public synchronized " +
IDLEntities.getBasicDataTypeJavaName(operation.retParam.type) + " " +
operation.operationID;
20       operBodyStart = "\t\t" +
IDLEntities.getBasicDataTypeJavaName(operation.retParam.type) + "
_retVal = " +
IDLEntities.getBasicDataTypeJavaInitValue(operation.retParam.type) +
";";
21       isOperations = "\t\t\t" + "_retVal = _inStream." +
IDLEntities.getBasicDataTypeReadFromStreamValue(operation.retParam.ty
pe) + "();";
22       hasReturnVal = true;
```

```
23     }
24     else
25     if (operation.retParam.type == 1)
26     {
27        operStart = "\t" + "public synchronized " + "void" + " " +
operation.operationID;
28     }
29     ...
```

I.4 Sekuencë shkrimi dhe leximi i parametrave në metodën processStandardOperation të klasës CodeGeneratorJacORBBindServer2

```
 1 while(stopPrm != null)
 2 {
 3    switch (stopPrm.direction){
 4       case 1:
 5          ...
 6       case 2:
 7          ...
 8       case 3:
 9          osSize += stopPrm.size;
10          isSize += stopPrm.size;
11          prmCount++;
12
13          if(prmCount == 1)
14          {
15             prmArgs =
IDLEntities.getBasicDataTypeJacORBHolderName(stopPrm.type) + " " +
stopPrm.name;
16          }
17          else if (prmCount > 1)
18          {
19             prmArgs += ", " +
IDLEntities.getBasicDataTypeJacORBHolderName(stopPrm.type) + " " +
stopPrm.name;
20          }
21
22          osOperations += "\r\n\t\t\t" + "_outStream." +
IDLEntities.getBasicDataTypeWriteToStreamValue(stopPrm.type) + "(" +
stopPrm.name + ".value);";
23          isOperations += "\r\n\t\t\t" + stopPrm.name + ".value =
_inStream." +
IDLEntities.getBasicDataTypeReadFromStreamValue(stopPrm.type) +
"();";
24          break;
25    }
26
27    stopPrm = stopPrm.next;
28 }
```

I.5 Sekuencë nga gjenerimi i kodit të gabimit në metodën processStandardOperation të klasës CodeGeneratorJacORBBindServer2

```
 1 xpcCount++;
 2 if (xpcCount == 1)
 3    throwExcpList = " throws " + (xptn.excptID > 128 ?
intrfc.interfaceName + "Package." + xptn.excptName : xptn.excptName);
 4 else
 5    throwExcpList += ", " + (xptn.excptID > 128 ?
intrfc.interfaceName + "Package." + xptn.excptName : xptn.excptName);
 6 String throwPart = "\r\n" + "\t\t\t" + "throw new " +
(xptn.excptID > 128 ? intrfc.interfaceName + "Package." +
xptn.excptName : xptn.excptName) + "(";
 7 int xpPrmTotSize = 0;
 8 if (xptn.firstParam != null)
 9 {
10    int xpPrmCount = 0;
11    Parameter xpPrm = xptn.firstParam;
12    while (xpPrm != null)
13    {
14       xpPrmCount++;
15       xpPrmTotSize+=xpPrm.size;
16       if (xpPrmCount == 1)
17          throwPart += "_inStream." +
IDLEntities.getBasicDataTypeReadFromStreamValue(xpPrm.type) + "()";
18       else
19          throwPart += ", " + "_inStream." +
IDLEntities.getBasicDataTypeReadFromStreamValue(xpPrm.type) + "()";
20       xpPrm = xpPrm.next;
21    }
22 }
23 throwPart += ");";
24 ifThrowCont += "\r\n" + "\t\t" + "else if " + (secondParenthGen ?

"(" : "") + "(_readStreamLength == (byte) " + (isHeaderSize +

xpPrmTotSize) + ")" + intrfcIDCheck + OpIDCheck +

(operation.generateStatusID ? " && (_readStatusID == (byte) " +

xptn.excptID + ")" : "") + (secondParenthGen ? ")" : "") + throwPart;
```

I.6 Sekuencë nga gjenerimi i operacionit brenda klasës së serverit lidhës në metodën processStandardOperation të klasës CodeGeneratorJacORBBindServer2

```
 1 pw.println(operStart + "(" + prmArgs  + ")" + throwExcpList);
 2 pw.println("\t{");
 3 if (hasReturnVal)
 4    pw.println(operBodyStart);
 5 if (intrfc.generateOperationID)
 6    pw.println("\t\t" + "byte _operationID = (byte) " +
this.currentOperationID + ";");
 7 pw.println("\t\t" + "byte _osSize = (byte) " + osSize + ";");
 8 pw.println("\t\t" + "byte _isSize = (byte) " + isSize + ";");
 9 pw.println("");
10 writeConnect(hasReturnVal);
11 ...
12 if (this.generateInterfaceID)
13    pw.println("\t\t\t" + "_outStream.writeByte(_interfaceID);");
14 if (intrfc.generateOperationID)
15    pw.println("\t\t\t" + "_outStream.writeByte(_operationID);");
16 if (osOperations.trim().length() > 0)
17    pw.println(osOperations.startsWith("\r\n") ?
osOperations.substring(2) : osOperations);
18 pw.println("\t\t\t" + "_ms.isDataReady = false;");
19 ...
20 writeDisconnect();
21 pw.println("");
22 pw.println("\t\t" + "StreamOperations2 _inStream = new
StreamOperations2(_ms.readStream);");
23 pw.println("\t\t" + "byte _readStreamLength =
_inStream.readByte();");
24 if (this.generateInterfaceID)
25    pw.println("\t\t" + "byte _readInterfaceID =
_inStream.readByte();");
26 ...
27 pw.println("\t\t" + "if " + (secondParenthGen ? "(" : "") +
"(_readStreamLength == _isSize)" + intrfcIDCheck + OpIDCheck +
(operation.generateStatusID ? " && (_readStatusID == (byte) 0)" : "")
+ (secondParenthGen ? ")" : "") + (isOperations.trim().length() == 0
? ";" : ""));
28 if (isOperations.trim().length() > 0)
29 {
30    pw.println("\t\t" + "{");
31    pw.println(isOperations.startsWith("\r\n") ?
isOperations.substring(2) : isOperations);
32    pw.println("\t\t" + "}");
33 }
34 if (ifThrowCont.trim().length() > 0)
35    pw.println(ifThrowCont.startsWith("\r\n") ?
ifThrowCont.substring(2) : ifThrowCont);
36 ...
```

```
37 pw.println("\t" + "}");
38 this.currentOperationID++;
```

SHTOJCA J

Kodi i gjeneratorit të serverit lidhës për Shërbime Ueb

J.1 Pjesë kodi nga gjenerimi i ndërfaqeve nga metoda generate e klasës CodeGeneratorJavaWSBindServer2

```
 1 ...
 2 ipw.println("@WebService(targetNamespace=\"" + intfc.interfaceName
+ "\")");
 3 ipw.println("@SOAPBinding(style=Style.RPC)");
 4 ipw.println("public interface " + intfc.interfaceName);
 5 ipw.println("{");
 6 InterfaceElement iel = (InterfaceElement) intfc.firstie;
 7 while(iel != null)
 8 {
 9    switch(iel.type)
10    {
11       case 1:
12          {
13             ipw.println("\t@WebMethod");
14             IEStandardOperation operation = (IEStandardOperation)
iel.object;
15             Parameter stopPrm = operation.firstParam;
16             String operStart = "";
17             String prmArgs = "";
18             int prmCount = 0;
19
20             if (operation.retParam.type > 1)
21             {
22                operStart = "\t" + "public " +
IDLEntities.getBasicDataTypeJavaName(operation.retParam.type) + " " +
operation.operationID;
23             }
24             else
25             if (operation.retParam.type == 1)
26             {
27                operStart = "\t" + "public " + "void" + " " +
operation.operationID;
28             }
29
30             while(stopPrm != null)
31             {
32                switch (stopPrm.direction){
33                   ...
34                   case 2:
35                      prmCount++;
36
37                      if(prmCount == 1)
```

```
38                      {
39                         prmArgs = "@WebParam(name = \"" +
stopPrm.name + "\", mode = WebParam.Mode.OUT) Holder<" +
IDLEntities.getBasicDataTypeJavaWSHolderNames(stopPrm.type) + "> " +
stopPrm.name;
40                      }
41                      else if (prmCount > 1)
42                      {
43                         prmArgs += ", " + "@WebParam(name = \"" +
stopPrm.name + "\", mode = WebParam.Mode.OUT) Holder<" +
IDLEntities.getBasicDataTypeJavaWSHolderNames(stopPrm.type) + "> " +
stopPrm.name;
44                      }
45
46                      break;
47                  ...
48                  }
49              }
50
51              IDLEExceptionList el = operation.firstExcptn;
52              while (el != null)
53              {
54                 IDLEException xptn = el.exception;
55                 if (xptn != null)
56                 {
57                    xpcCount++;
58                    if (xpcCount == 1)
59                       throwExcpList = " throws " + (xptn.excptID >
128 ? intfc.interfaceName + "Package." + xptn.excptName +
"_Exception" : xptn.excptName + "_Exception");
60                    else
61                       throwExcpList += ", " + (xptn.excptID > 128 ?
intfc.interfaceName + "Package." + xptn.excptName + "_Exception" :
xptn.excptName + "_Exception");
62                 }
63                 el = el.next;
64              }
65           }
66        ...
67        }
68 }
69 ...
```

J.2 Pjesë kodi nga gjenerimi i klasëve të gabimeve dhe fault nga metoda generateException e klasës CodeGeneratorJavaWSBindServer2

```
 1 private void generateException(IDLEException excptn, String
interfaceName) throws IOException
 2 {
 3    String newFilePath = this.filePath;
 4    boolean intrfcNotEmpty = interfaceName.trim().length() > 0;
```

```
 5     if (intrfcNotEmpty)
 6     {
 7        newFilePath += "\\" + interfaceName + "Package";
 8        File theExcDir = new File(newFilePath);
 9
10        if (!theExcDir.exists()) {
11           ...
12        }
13     }
14
15     PrintWriter epw = new PrintWriter(new FileWriter(newFilePath +
"\\" + excptn.excptName + ".java"));
16     if (intrfcNotEmpty)
17     {
18        epw.println("package " + interfaceName + "Package;");
19        epw.println();
20     }
21     epw.println("import java.lang.Exception;");
22     epw.println();
23     epw.println("public class " + excptn.excptName + " extends
Exception {");
24
25     Parameter excptnPrm = excptn.firstParam;
26     ...
27     while(excptnPrm != null)
28     {
29        ...
30     }
31
32     if (prmCount > 0) epw.println(prmGlobal);
33     epw.println("\tpublic " + excptn.excptName + "(" + prmArgs + ")
{");
34     if (prmCount > 0) epw.println("\t\t" + prmThis.trim());
35     epw.println("\t}");
36     epw.print("}");
37     epw.close();
38
39     PrintWriter fpw = new PrintWriter(new FileWriter(newFilePath +
"\\" + excptn.excptName + "_Exception.java"));
40     if (intrfcNotEmpty)
41     {
42        fpw.println("package " + interfaceName + "Package;");
43        fpw.println();
44     }
45     fpw.println("import javax.xml.ws.WebFault;");
46     fpw.println();
47     fpw.println("@WebFault(name = \"" + excptn.excptName + "\",
targetNamespace = \"" + interfaceName + "\")");
48     fpw.println("public class " + excptn.excptName + "_Exception
extends Exception {");
49     fpw.println("\tprivate " + excptn.excptName + " faultInfo;");
50     fpw.println();
51     ...
52     fpw.println("\tpublic " + excptn.excptName + "_Exception(String
message, " + excptn.excptName + " faultInfo, Throwable cause) {");
53     fpw.println("\t\tsuper(message, cause);");
54     fpw.println("\t\tthis.faultInfo = faultInfo;");
55     fpw.println("\t}");
```

```
56      fpw.println();
57      fpw.println("\tpublic " + excptn.excptName + " getFaultInfo()
{");
58      fpw.println("\t\treturn faultInfo;");
59      fpw.println("\t}");
60      fpw.print("}");
61      fpw.close();
62  }
```

SHTOJCA K

Kodi i shembullit të sistemit për marrje të dhënash

K.1 Sekuencë nga implementimi i operacionit readAD të ndërfaqes ADReader

```
 1 IntImpl_ADReader_0_STOP_readAD_ID:
 2 ; Please find in and inout parameters at RAM locations as follows:
 3 ; parameter, name: channelID, direction: in, CORBA type: octet,
size: 1, RAM location: 129
 4 ; In case your operation terminates successfully ...
 5 ; Please place return, out and inout parameters at RAM locations
as follows:
 6 ; parameter, direction: return, CORBA type: short, size: 2, RAM
locations: 128..129
 7 ; Your code for readAD from here
 8 ; Uncomment the following code to terminate sucessfully at the end
of your code
 9 ; MOV A, #0
10 ; JMP DcdRspns
11 ; In case the operation terminates with ChannelOutOfRange
exception ...
12 ; Please place param(s) at following RAM addresses:
13 ; parameter, name: minChannelID, size: 1, RAM location: 128
14 ; parameter, name: maxChannelID, size: 1, RAM location: 129
15 ; Uncomment the following code to terminate with exception
ChannelOutOfRange (ID: 1) at the end of your code
16 ; MOV A, #1
17 ; JMP DcdRspns
18 ; In case the operation terminates with ChannelNotInUse exception
...
19 ; Uncomment the following code to terminate with exception
ChannelNotInUse (ID: 2) at the end of your code
20 ; MOV A, #2
21 ; JMP DcdRspns
22 ; ******* WRITE CODE HERE *******
23 MOV R0, #129
24 MOV A, @R0
25 CJNE A, #0, ChkIfID0Failed
26 JMP ImplID0
27 ChkIfID0Failed:
28 JMP ChkIfID1
29 ImplID0:
30 ; Read from ADC MCP3204C
31 ADC_CS EQU P3.5
32 ADC_CLK EQU P1.7
33 ADC_DIN EQU P1.5
34 ADC_DOUT EQU P1.6
35 ...
```

```
36 ; END reading from ADC MCP3204C
37 MOV A, #0
38 JMP DcdRspns
39 ChkIfID1:
40 CJNE A, #1, ChkIfID2
41 MOV A, #2
42 JMP DcdRspns
43 ChkIfID2:
44 CJNE A, #2, ChkIfID3
45 MOV A, #2
46 JMP DcdRspns
47 ChkIfID3:
48 CJNE A, #3, ChkIfIDOther
49 MOV A, #2
50 JMP DcdRspns
51 ChkIfIDOther:
52 MOV R0, #128
53 MOV @R0, #0
54 INC R0
55 MOV @R0, #3
56 MOV A, #1
57 JMP DcdRspns
58 ; End of your code for readAD
```

K.2 Përkufizimet IDL të serverit DAQLoop - serveri

```
#include "sample.idl"

interface DAQLoop
{
    void run();
    void runWithPeriod(in unsigned long long period, in unsigned long
long delay);
    void stop();

    attribute unsigned short samplesID;
    readonly attribute boolean isRunning;
    readonly attribute Sample getLastSampleIn;
    readonly attribute unsigned long long getPeriod;

    void loadSamplesDetails();
    void loadTransformation();
    void loadAlarms();
    void loadAll();
};
```

K.3 Përkufizimet IDL të serverit DAQLoop - skedari sample.idl

```
struct Sample
{
    unsigned long long smplNo;
    unsigned short smplsID;
    string  DateAndTime;
    unsigned long origBytes;
    double calcValue;
};
```

K.4 Sekuencë nga implementimi i serverit DAQLoop

```
 1 public void run()
 2 {
 3    if(timer != null)
 4       timer.cancel();
 5    timer = new java.util.Timer(true);
 6    timer.scheduleAtFixedRate(new java.util.TimerTask()
 7    {
 8       public void run()
 9       {
10          executeTasks();
11       }
12    },
13    0, timerPeriod);
14    this.isRunning = true;
15 }
16
17 private void executeTasks()
18 {
19    try
20    {
21       Sample smpl = new Sample();
22       smpl.origBytes = adReader.readAD((byte) 0);
23       java.util.Date now = new java.util.Date();
24       smpl.DateAndTime = (now.getYear()+1900)+"-
"+(now.getMonth()+1)+"-"+now.getDate()+"
"+now.getHours()+":"+now.getMinutes()+":"+now.getSeconds();
25       smpl.smplsID = this.samplesID;
26       if(transformation.transformationType == 0)
27       {
28          smpl.calcValue = 0.0;
29          smpl.smplNo = dbStoringSamples.insertSample2(smpl);
30       }
31       else
32       {
33          smpl.calcValue =
transformation.coefficientA*Math.pow(((double)smpl.origBytes),3)+tran
sformation.coefficientB*Math.pow(((double)smpl.origBytes),2)+transfor
mation.coefficientC*((double)smpl.origBytes)+transformation.coefficie
ntD;
34          smpl.smplNo = dbStoringSamples.insertSample(smpl);
35       }
36
```

```
37          ...
38
39      }
40      catch (Exception xps)
41      {
42          System.out.println(xps);
43      }
44  }
45
46  public void stop()
47  {
48      if(timer != null)
49      {
50          timer.cancel();
51          timer = null;
52      }
53
54      this.isRunning = false;
55  }
```