UNIVERSITATEA TEHNICĂ A MOLDOVEI

Cu titlu de manuscris
C. Z. U: 621.38:004.75:681.5(043)

BRAGARENCO ANDREI

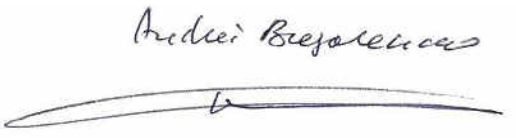
MODELAREA SISTEMELOR ELECTRONICE DISTRIBUITE CU GENERAREA AUTOMATĂ A CONFIGURAȚIEI

122.03 – MODELARE, METODE MATEMATICE, PRODUSE PROGRAM

Teză de doctor în informatică

Conducător științific: Marusic Galina,
dr., conf. univ.

Autor: Bragarenco Andrei



CHIŞINĂU, 2023

CUPRINS

ADNOTARE

la teza „Modelarea sistemelor electronice distribuite cu generarea automată a configurației” prezentată de către Bragarenco Andrei pentru conferirea titlului științific de doctor în informatică, Chișinău, 2023.

Structura tezei: introducere, 3 capitole, concluzii, bibliografie – 179 titluri, 10 anexe, 155 de pagini text de bază, inclusiv 118 figuri și 3 tabele. Rezultatele sunt publicate în 13 lucrări.

Cuvinte-cheie: modelare; sistem electronic distribuit; Internetul lucrurilor (IoT); stivă de componente; arhitectura în straturi; condiționare semnal; lanț de comunicare; generator cod; metamodel; automatizare.

Domeniul de studiu: produse program, modelarea sistemelor electronice distribuite, generarea automată a resurselor de program.

Scopul tezei: proiectarea sistemelor electronice distribuite cu generarea automată a configurației în baza dezvoltării de modele și produse program, utilizând concepte arhitecturale în straturi și metode de organizare a fluxurilor de informație în lanțuri de comunicare.

Obiective: studiul abordărilor actuale cu privire la modelarea sistemelor electronice distribuite; elaborarea conceptului de *arhitectură generică* a unui sistem electronic incorporat; elaborarea unui *concept comun pentru achiziția informației și acționare* asupra mediului; elaborarea unui concept pentru *organizarea fluxurilor de informație*; definirea *modelului de componentă configurabilă* a sistemului; elaborarea metodologiei de modelare a sistemelor electronice distribuite; dezvoltarea *produsului program* pentru modelare și *definirea automată a configurațiilor*; crearea resurselor online de *componente reutilizabile*; dezvoltarea aplicaților prin metoda propusă; analiza rezultatelor obținute.

Noutatea și originalitatea științifică: a fost definit un concept generic de arhitectură a unui sistem incorporat. S-au evidențiat modurile de interacțiune cu mediul și similaritatea între condiționarea semnalelor în fluxurile de intrare cu cele de ieșire. A fost elaborat un concept de *lanț de comunicare* comun pentru interacțiunile senzor-actuator și comunicarea cu alte dispozitive. Rezultatele cercetării permit elaborarea unei platforme de modelare a sistemelor electronice distribuite în baza modelelor *arhitecturale în straturi* și metode de *organizare a lanțurilor de comunicare*.

Problema științifică soluționată constă în dezvoltarea de sisteme electronice distribuite în baza modelării de sistem, produselor program și metodelor moderne de descriere a sistemelor prin metamodele pentru arhitecturi în straturi și gestionare a fluxurilor de informații cu lanțuri de comunicare, fapt care a condus la automatizarea procesului de elaborare, configurare și generare de cod sursă de platformă, ceea ce a permis optimizarea timpului alocat modelarii de sisteme și funcționalităților esențiale a acestora.

Semnificația teoretică: în baza modelelor conceptuale de *arhitectură în straturi* și *lanțuri de comunicare* a fost elaborata o metodologie de dezvoltare a platformelor de resurse programabile pentru sisteme electronice distribuite.

Valoarea aplicativă: în premieră, au fost construite modele de sisteme electronice distribuite, făcându-se abstracție de mediul de propagare a semnalului, acestea fiind abstractizate prin utilizarea lanțurilor de comunicare, producând interfețe între funcțiile de transfer ale sistemului. Rezultatele cercetării pot fi aplicate pentru elaborarea platformelor de sisteme electronice distribuite în diverse domenii.

Implementarea rezultatelor științifice a avut loc în cadrul companiei „Viomarix-Plus” SRL; A.O. „Asociația Artelor Alternative ARTWATT” Rep. Moldova; companiei „Arobs Software” SRL, Rep. Moldova; companiei „ABAS-NT” Franța; programelor de cercetare și dezvoltare din cadrul Universității Tehnice a Moldovei, precum și în diverse proiecte individuale.

ABSTRACT

to thesis "Modeling of distributed electronic systems with automatic configuration generation", presented by Bragarenco Andrei for the conferral of the scientific title of doctor in Informatics, Chișinău, 2023.

The thesis structure: introduction, 3 chapters, conclusions, bibliography with 179 titles, 10 appendices, 155 pages of basic text, including 118 figures and 3 tables. The results are published in 13 papers.

Keywords: modeling; distributed electronic system; Internet of Things (IoT); component stack; layered architecture; signal conditioning; communication chain; code generator; metamodel; automation.

The field of the investigation: software products, modeling of distributed electronic systems, automatic generation of program resources.

The thesis aim: design of distributed electronic systems with automatic configuration generation based on the development of models and software products, using layered architectural concepts and methods for organizing information flows in communication chains.

The objectives: study of current approaches regarding the modeling of distributed electronic systems; elaboration of the generic architecture concept of an embedded electronic system; development of a common concept for the acquisition of information and action on the environment; development of a concept for organizing information flows; defining the configurable system component model; development of the modeling methodology of distributed electronic systems; development of the software product for modeling and automatic definition of configurations; creating online resources of reusable components; development of applications using the proposed method; analysis of the results obtained.

Scientific novelty and originality of the results: A generic concept of an embedded system architecture has been defined. The modes of interaction with the environment and the similarity between the conditioning of signals in the input and output flows have been highlighted. A common communication chain concept has been developed for sensor-actuator interactions and communication with other devices. The research results allow the development of a platform for modeling distributed electronic systems based on layered architectural models and methods for organizing communication chains.

The scientific problem solved consists in the development of distributed electronic systems based on system modeling, software products and modern methods of system design through metamodels for layered architectures and management of information flows with communication chains, which led to the automation of the process of development, configuration and platform source code generation, which allowed to optimize the time allocated to the modeling of systems and their essential functionalities.

The theoretical importance: based on the conceptual models of architecture in layers and communication chains, a methodology for the development of programmable resource platforms for distributed electronic systems was developed.

The applied value of the thesis: for the first time, models of distributed electronic systems have been built, abstracting from the domain of signal propagation, these being abstracted through the use of communication chains, producing interfaces between the transfer functions of the system. The research results can be applied to the development of distributed electronic systems platforms in various fields.

The scientific results implementation took place within the company "Viomarix-Plus" SRL; A.O. “Association of Alternative Arts ARTWATT" Rep. Moldova; of the company “Arobs Software” SRL, Rep. Moldova; of the company “ABAS-NT” France; research and development programs within the Technical University of Moldova, as well as in various individual projects.

АННОТАЦИЯ

к диссертации «Моделирование распределенных электронных систем с автоматической генерацией конфигурации», представленной Брагаренко Андрей на соискание ученой степени доктора наук в информатики, Кишинев, 2023.

Структура диссертации: введение, 3 главы, заключение, библиография – 179 названий, 10 приложений, 155 страниц основного текста, в том числе 118 рисунков и 3 таблицы. Результаты опубликованы в 13 статьях.

Ключевые слова: моделирование; распределенная электронная система; Интернет вещей (IoT); стек компонентов; многоуровневая архитектура; преобразование сигнала; цепь связи; генератор кода; метамодель; автоматизация.

Область исследования: программные продукты, моделирование распределенных электронных систем, автоматическая генерация программных ресурсов.

Цель диссертации: проектирование распределенных электронных систем с автоматическим формированием конфигурации на основе разработки моделей и программных продуктов, с использованием многоуровневых архитектурных решений и методов организации информационных потоков с цепочками связи.

Задачи работы: изучение современных подходов к моделированию распределенных электронных систем; разработка общей концепции архитектуры встроенной электронной системы; разработка общей концепции получения информации и воздействия на окружающую среду; разработка концепции организации информационных потоков; определение модели конфигурируемого компонента системы; разработка методологии моделирования распределенных электронных систем; разработка программного продукта для моделирования и автоматического определения конфигураций; создание онлайн-ресурсов повторно используемых компонентов; разработка приложений с помощью предлагаемого метода; анализ полученных результатов.

Научная новизна и оригинальность результатов: определено общее понятие архитектуры встраиваемой системы. Выделены режимы взаимодействия с окружающей средой и сходство между обработкой сигналов во входном и выходном потоках. Для взаимодействия датчика и исполнительного механизма и связи с другими устройствами была разработана общая концепция цепи связи. Результаты исследований позволяют разработать платформу для моделирования распределенных электронных систем на основе многоуровневых архитектурных моделей и методов организации коммуникационных цепочек.

Решенная научная проблема заключается в разработке распределенных электронных систем на основе системного моделирования, программных продуктов и современных методов описания систем посредством метамоделей для многоуровневых архитектур и управления информационными потоками с цепочками связи, что привело к автоматизации процесса разработки, настройки и генерации исходного кода платформы, что позволило оптимизировать время, отведенное на моделирование систем и их основных функций.

Теоретическая значимость работы: на основе концептуальных моделей архитектуры слоев и коммуникационных цепочек разработана методология разработки программируемых ресурсных платформ для распределенных электронных систем.

Практическая значимость работы: впервые были построены модели распределенных электронных систем, абстрагирующихся от среды распространения сигналов, абстрагирующихся за счет использования коммуникационных цепей, создающих интерфейсы между передаточными функциями системы. Результаты исследования могут быть применены при разработке платформ распределенных электронных систем в различных областях.

Внедрение научных результатов происходило в рамках компании "Viomarix-Plus" SRL; АО Ассоциация альтернативных искусств "ARTWATT" Респ. Молдова; компании Arobs Software SRL, респ. Молдова; компания ABAS-NT Франция; программы исследований и разработок в рамках Технического Университета Молдовы, а также в различных индивидуальных проектах.

LISTA ABREVIERILOR

ACT – ACTuator (Componenta de Acționare)

ADC – Analog Digital Convertor (Convertor Analog Digital)

AI – Artificial Intelligence (Inteligență Artificială)

API – Application Programming Interface (Interfață de Programare a Aplicațiilor)

ARM – Advanced Risk Machine (Denumire arhitectura microcontroller)

ASW – Application SoftWare (Software de aplicație)

AVR – Advanced Virtual RISC (arhitectură RISC virtuală avansată)

AUTOSAR – AUTomotive Open System ARchitecture (Arhitectură de Sistem Deschis Automotive)

BMT – Behavioral Modelling Tool (Instrument de Modelare Comportamentală)

BSW – Basic SoftWare (Software de bază)

CAD – Computer Aided Design (Proiectare Asistată de Calculator)

CAN – Controller Area Network (Rețea de Controllere)

CDD – Complex Device Driver (Driver de Dispozitiv Complex)

COM – COMmunication (Componentă de Comunicare)

CNC – Control Numeric Computerizat

DAC – Digital Analog Convertor (Convertor Digital Analogic)

DOF – Degree of Freedom (Grad de Libertate)

DB – Data Base (Bază de Date)

DIO – Digital Input Output (Intrare Ieșire Digitală)

DSML – Domain Specific Modeling Languages (Limbaje de Modelare Specifice Domeniului)

ECG – ElectroCardioGramă

ECU – Electronic Control Unit (Unitate de Control Electronic)

ECAL – ECU Abstraction Layer (Stratul de Abstractizare ECU)

EE – Electronic Engineering (Inginerie Electronică)

EEPROM – Electrically Erasable Programmable Read-Only Memory (Memorie Programabilă cu Drepturi Doar de Citire cu Ștergere Electrică)

ERP – Enterprise Resource Planning (Sistem de Planificare a Resurselor Întreprinderii)

ESW – Extended SoftWare (Software Extins)

FPGA – Field Programmable Gate Array (Matrice de Porți Logice Programabilă)

GPIO – General Purpose Input Output (Intrare-Ieșire de Uz General)

GPS – Glogal Positioning System (Sistem de Poziționare Globală)

HMI – Human Machine Interface (Interfața Om-Mașină)

HTTPS – HyperText Transfer Protocol Secure (Protocol de Transfer HypertText Securizat)

HW – Hardware (Produs Electronic)

IDC – International Data Corporation (Denumire companie)

IDE – Integrated Development Environment (Mediu de Dezvoltare Integrat)

IEEE – Institute of Electrical and Electronics Engineers (Institutul de Inginerie Electronică și Electrică)

I2C – Inter Integrated Circuit Protocol (Protocol între Circuite integrate)

IIoT – Industrial Internet of Things (Internetul Industrial al Lucrurilor)

IO – Input-Output interface (Interfață de Intrare-Ieșire)

IoT – Internet of Things (Internetul Lucrurilor)

IP – Internet Protocol

IPC – Inter Process Communication (Comunicare Între Procese)

IT – Informational Technology (Tehnologie Informațională)

ITT – Institute of Technology Tralee (Institutul de Tehnologie din Tralee, Irlanda)

JSON – JavaScript Object Notation (Notatie de Obiecte JavaScrit)

LAN – Local Area Network (Rețea Locală)

LoWPAN – Low-Power Wireless Personal Area Networks (Rețele Personale Fără Fir de Putere Mică)

LoRaWAN – Lo(ng) Ra(nge) Wide Area Network (Rețea Extinsă de Distanță Mare)

FTJ – Filtru Trece Jos

LPWAN – Low Power Wide Area Networks (Rețele extinse de putere redusă)

M2M – Machine to Machine (Mașină la Mașina)

M2P – Machine to People (Mașină pentru Om)

MCU – Micro Controller Unit (Microcontroller)

MDE – Model Driven Engineering (Ingineria Bazată pe Modele)

ME – Mechanical Engineering (Inginerie Mecanică)

MEM – MEMory (Memorie)

MEMS – Micro Electro Mechanic System (Sisteme Micro Electro Mecanice)

MQTT - Message Queuing Telemetry Transpor (Protocol de Transport al Mesajelor de Telemetrie)

MIT – Massachusetts Institute of Technology

NFC – Near Field Communication (Comunicare în Câmp Apropiat)

OECD – Organisation for Economic Co-operation and Development (denumire organizatie)

OEM – Original Equipment Manufacturer (Producător Original de Echipamente)

OS – Operating System (Sistem de Operare)

OSI – Open Systems Interconnection (Interconexiuni Sisteme - Resurse Deschise)

P2P – People to People (Persoană la Persoană)

PAN – Personal Area Network (Rețele Personale)

PC – Personal Computer (Calculator Personal)

PHY – PHYsical environment (mediu fizic/înconjurător)
POW – Componenta de POWer Management (Gestionarea Energiei)
QoS – Quality of Service (Calitatea Serviciului)
QR – Quick Response code (Cod de Răspuns Rapid)
REST – Representational State Transfer (Transfer de Stat Reprezentativ)
RFID – Radio Frequency Identification (Identificarea prin Frecvențe Radio)
ROS – Robot Operating System (Sistem de Operare Robot)
ROSCORE – ROS Core (Serviciu Central ROS)
RTE – RunTime Environment (Mediu de Rulare)
SOA – Service Oriented Architecture (Arhitectură Orientată pe Servicii)
SoC – System-on-Chip (Sistem într-un singur Cip).
SNS – SeNSor (componentă tip senzor)
SNMP – Simple Network Management Protocol (Protocol simplu de gestionare a rețelei)
SPI – Serial Peripheral Interface (Interfața periferică serială)
SRVL – Service Layer (Stratul de Servicii)
SSH – Secured Shell (Terminal Securizat)
SW – SoftWare (Produs Program)
SWC – SoftWare Component (Componente Software)
SWE – SoftWare Engineering (Inginerie Software)
TCP – Transmission Control Protocol (Protocol de control al transferului)
TMDA – Time Division Multiple Access (Acces multiplu divizat în timp)
UE – Uniunea Europeană
UDP – User datagram protocol (Protocol de date de utilizator)
UI – User Interaction (Interacțiune cu Utilizatorul)
USART – Universal Snchronous Asynchronous Receiver Transmiter (Interfața Sincronă-Asincronă de Recepție-Transmisie)
ULSI – Ultra Large Scale Integration (Integrare pe Scara Ultra Mare)
VFB - Virtual Functional Bus (Magistrala Virtuală de Funcționare)
VLSI – Very Large Scale Integration (Integrare pe Scara Foarte Mare)
WDT – WatchDog Timer (Cronometru de Supraveghere)
Wi-Fi – Wireless Internet Protocol (Protocol de Rețea Fără Fir)
WSN – Wireless Sensor Network (Rețea Fără Fir de Senzori)
WPAN – Wireless Personal Area Networks (Rețele Personale Fără Fir)
WWAN – Wireless wide area network (Rețea Extinsă Fără Fir)
XML – eXtensible Markup Language (Meta-Limbaj de Marcare)

INTRODUCERE

Actualitatea temei și importanța problemei abordate

Dezvoltările recente în software, hardware și tehnologii de comunicare au dus la o creștere bruscă a lucrurilor conectate la Internet. Numărul dispozitivelor conectate la Internet a crescut exponențial, la fel și tipul acestora – dispozitivele care până acum nu erau gândite sunt acum participanți la ecosistemul Internet of Things / Everything. Reutilizarea soluțiilor Internet of Things (IoT), numită și Internetul Lucrurilor, existente sau a componentelor soluției este crucială pentru a face față cererii mari de noi tipuri de lucruri care să fie conectate la rețea. Este crucială și o abordare arhitecturală bine definită, care implementează reutilizarea.

Dezvoltarea tehnologiilor microelectronice pun la dispoziție tot mai multe soluții de sisteme într-un chip cu putere de procesare în creștere, fapt care a permis multiplicarea într-un tempou sporit al numărului de dispozitive încorporate. Un mare avantaj îl aduc platformele de dezvoltare de proiecte prin configurare și generare de cod, contribuind la reducerea efortului de adaptare a resurselor reutilizabile în cadrul proiectului și configurarea formală a acestora. În urma utilizării unor asemenea tehnologii de automatizare, efortul de dezvoltare revine satisfacerii cerințelor funcționale ale aplicației de nivel înalt, descriptiv, prin utilizare de metode formale sau metalimbaje.

Comunicarea este actul de a transfera informații între entități. Există multe abordări de comunicare arhitecturală care oferă soluții pentru transferul de date de la transmițător la receptor. Chiar și acele soluții vizează domenii diferite, o mulțime de asemănări ar putea fi identificate în arhitecturile lor.

Odată cu dezvoltarea tehnologiilor și a nivelului sporit de integrare a funcționalităților pe unitate de echipament, creșterea diversității și a numărului dispozitivelor interconectate, iese în evidență problema controlului dimensiunii și complexității sistemelor interconectate în ecosistemul Internet of Things. Totodată, cererea actuală a pieței pentru soluții cu o diversitate largă de aplicații pune în evidență problema de constrângere a timpului de realizare și adaptare a soluțiilor la cerințele specifice aplicației.

Din punctul de vedere al tehnologiei, în diverse sectoare, cum ar fi producția industrială, controlul mediului și al microclimei, sectorul agricol, logistica și marketingul, există abordări comune ale digitalizării. Conform abordării arhitecturale generice, propusă în teză, pentru un sistem electronic distribuit, acesta constă din tehnologii specifice domeniului de aplicare. Urmând conceptul IoT ca Internet al Orice, componentele sistemului pot fi grupate, după problemele pe care le soluționează, cum ar fi:

Achiziție și diagnosticare de date (SNS) – acest domeniu constă în instrumente și tehnici de achiziție de date primare de mediu pentru monitorizarea și prelucrarea în teren în vederea detectării unor aspecte specifice de interes. Accentul principal este pus pe senzorii specifici domeniului ca componente hardware pentru sisteme automate, dar ar putea fi luată în considerare și introducerea manuală a datelor despre mediu. Exemple de astfel de tehnologii ar putea fi stațiile meteorologice, achiziție informație despre temperatură și de umiditate a solului, inspecția cu drone, monitorizarea prin satelit sau chiar laborator manual de analize cu introducerea datelor prin interacțiunea cu utilizatorul.

Acționare și protecție (ACT) – este mai mult legată de lucrările de câmp și de prelucrare, unde protecția este mai mult legată de operațiunea agriculturii de precizie pentru a atinge doar ținta și pentru a nu afecta zonele nedorite sau pentru a exclude operațiunile inutile. Printre tehnologiile din acest grup se pot aminti industria de precizie, manipulare obiecte sau chiar agricultura de precizie. Tot aici ar putea fi incluși și roboții autonomi, combinele și tractoarele.

Interacțiunea cu utilizatorul (UI) – acest domeniu este legat mai mult de experiența utilizatorului de a gestiona datele, prezentarea informațiilor, accesibilitatea, interpretarea ușoară. În acest domeniu se pot include dispozitivele și aplicațiile mobile, portalurile de Internet, fluxurile de știri, sistemele de notificare.

Comunicații și rețele (COM) – includ instrumentele și tehnicile care interconectează tehnologiile. Cel mai reprezentativ din acest grup este accesul la Internet prin intermediul rețelelor cu fir sau fără fir, precum Ethernet, WiFi sau 3/4/5G. Rețelele de tip IoT de senzori/actuatori fiind cele mai potrivite pentru acoperire de terenuri cu întinderi mari, acestea fiind *State of The Art* (stadiul actual al tehnologiilor) pentru comunicarea între dispozitive. Una dintre cele mai populare tehnologii IoT este rețeaua LoRa WAN, care este foarte scalabilă, are un consum redus de energie, flexibilă și costuri reduse.

Stocarea informațiilor și Clouding – acest domeniu este legat de tehnologiile de stocare a datelor în care *State of The Art* este tehnologia Cloud, care este o bază pentru tehnologia big data, statistici, la fel ca și pentru tehnicile de flux de informații Highway.

Management și Control (Aplicație și Platformă) – acest domeniu este legat de aplicație și în principal de Aplicații SW și platformele pentru managementul informațiilor și resurselor. Rolul acestui domeniu este de a fi un integrator al tuturor subsistemelor menționate, care utilizează:

- date achiziționate în timp real din diferite surse prin domeniul SNS;
- date din rețea prin domeniul COM;
- date statistice stocate în Cloud.

Controlul se manifestă prin utilizarea tehnologiilor Bigdata sau Machine Learning, sau altele clasice, pentru a genera decizii de control a procesului care va fi aplicat mediului extern prin intermediul componentelor ACT. Aplicația oferă o interfață de utilizare UI îmbunătățită pentru o experiență mai bună pentru utilizator.

Cele mai avansate tehnologii sunt sistemele de tip ERP – Enterprise Resource Planning, care ar putea fi considerate tabloul de bord, oferind un grup de instrumente pentru gestionarea unei întreprinderi. Astfel de sisteme ar putea oferi soluții pentru optimizarea intrărilor, controlul eficient al proceselor, distribuția inteligentă și chiar recomandări de marketing.

Eficiență energetică (POW) – acest domeniu este legat de sustenabilitate. Energia folosită în producție joacă unul dintre cele mai mari roluri în prețul final al produsului, astfel ca optimizarea, reducerea sau chiar excluderea utilizării sursei externe de energie va reduce considerabil costul de producție. În acest grup de tehnologii ar putea fi clasificate: panoul de energie solară electrică și termică; eoliene; pompele de încălzire subterane; la fel și sistemele pasive, precum izolarea clădirilor.

Reieșind din importanța problemelor menționate, în teză se propune o generalizare a metodelor de interacțiune într-o arhitectură generică de sistem electronic încorporat și elaborarea unei metodologii de automatizare a procesului de proiectare prin componente configurabile și generare de cod de interacțiune între componente.

Scopul lucrării:

Proiectarea sistemelor electronice distribuite cu generarea automată a configurației în baza dezvoltării de modele și produse program, utilizând concepte *arhitecturale în straturi* și metode de *organizare a fluxurilor de informație* în lanțuri de comunicare, scop atins prin **următoarele obiective**:

1. Elaborarea conceptului de arhitectură generică a unui sistem electronic încorporat prin evidențierea domeniilor de interacțiune și definirea de componente în straturi.
2. Elaborarea unui concept comun pentru componentele de achiziție a informației și acționare asupra mediului, asociind diagnozele cu achiziția, iar reacțiile de protecție – cu acționarea.
3. Elaborarea unui concept pentru organizarea fluxurilor de informație a componentelor de interacțiune cu mediul și interacțiunea între dispozitivele electronice.
4. Definirea modelului de componentă configurabilă a sistemului, dar și a metodei de adaptare a soluțiilor externe preluate și reutilizate în cadrul sistemului.
5. Elaborarea metodologiei de modelare a sistemelor electronice distribuite prin *modelare arhitecturală* cu metamodele și generarea automată a configurației și a codului sursă.

6. Elaborarea produsului program pentru modelarea și definirea automată a configurațiilor bazată pe metamodele și generare de cod sursă de platformă.
7. Crearea resurselor online de componente reutilizabile ca bază de soluții pentru dezvoltare de sisteme, în scopul optimizării timpului alocat modelarii de sisteme în baza de resurse existente.
8. Dezvoltarea aplicaților prin metoda de modelare a sistemelor electronice distribuite pentru validarea metodei propuse.
9. Analiza rezultatelor obținute cu scopul adaptării acestora la noile domenii de aplicații.

Ipoteza de cercetare are la bază faptul că procesul de comunicare reprezintă actul de transfer de informație între interlocutori, indiferent de forma acestora, fie ființă umană, mediu ambiant, sistem, dispozitiv, componentă electrică sau resursă program. Multitudinea de tehnologii cunoscute vin cu diverse soluții la diferite nivele de abstracție a procesului de comunicare, atât pentru accesarea și interacționarea cu date și informații din cadrul sistemelor informatice, cât și din exteriorul acestora. Acest fapt conduce la ideea definirii unei metode generale automatizate pentru stabilirea de interacțiuni pe toate nivelele de abstractizare. Ipoteza de cercetare se bazează pe următoarele componente:

- un sistem este definit de subsisteme ca un ansamblu de dispozitive electronice, formând împreună un sistem electronic distribuit;
- o anumită interacțiune a sistemului distribuit cu mediul extern poate fi realizată prin intermediul unui singur dispozitiv electronic, totodată poate fi distribuită între mai multe dispozitive ce formează sistemul;
- interacțiunile externe ale aplicațiilor din dispozitivele electronice se realizează cu modele arhitecturale în straturi, care abstractizează diferite nivele, cum ar fi parametrul mediului fizic, semnalul electric, interfața electrică-digitală, informația;
- interacțiunile între participanții la procesul de comunicare se definesc prin fluxuri complexe de informație, în funcție de mediile prin care trece informația respectivă și canalele de comunicare;
- fiecare element al fluxului de informație este definit ca o componentă configurabilă pentru facilitarea transformării informației reprezentată de un metamodel.

Sinteza metodologiei de cercetare și justificarea metodelor de cercetare alese

Metodele de cercetare utilizate în teză:

- Abstractizarea – au fost abstractizate: dispozitivele care fac parte din sistemul electronic distribuit, domeniile de interacțiune externă – mediu, utilizator echipamente; nivelele de interacțiune cu mediul, cum ar fi semnal, interfață electrica-digitală, informație; funcțiile de

transfer pentru elementele ce formează lanțurile de comunicare; canale de comunicare stabilite între participanții la o interacțiune.

- Formalizarea – s-au formalizat condițiile pe care trebuie să le îndeplinească o componentă pentru a accesa informații de la altă componentă, utilizându-se noțiunile de funcție de transfer, lanț de comunicare sau canal.
- Inducția – au fost făcute raționamente, de la particular la general, și anume: prin generalizare se permite ca modalitatea de interacțiune aplicată între două componente să fie aplicată în construirea de lanțuri complexe de comunicare pentru realizarea interacțiunilor, la fel ca și definirea funcționalităților sistemului în baza de componente cu funcții de transfer specifice cerințelor aplicației.
- Deducția – au fost făcute raționamente, de la general la particular, și anume:
    - de la sistemul electronic distribuit la dispozitivul electronic;
    - de la grupuri de componente la componentă;
    - de la domeniile de interacțiuni la domeniu;
    - de la componenta de interacțiune la nivelul de abstracție;
    - de la canalul de comunicare la lanțul de comunicare;
    - de la lanțul de comunicare la funcție de transfer.
- Clasificarea și tipologia – prin operația logică de divizare a volumului noțiunii, au fost definite clasificări ale interacțiunilor, după mai multe criterii: după modul de implementare (HW, SW); după complexitate (funcții de transfer și canal/lanț de comunicare); după nivele de abstracție (interfață HW-SW, componente electronice-driver, dispozitiv-serviciu); după domenii de interacțiune (mediu, utilizator, dispozitiv/rețea).
- Metoda axiomatică – prin utilizarea de enunțuri afirmative, cu definiții, cum ar fi: sistem electronic distribuit, dispozitiv electronic distribuit, arhitectura în straturi, nivel de abstracție, componentă, domeniu de interacțiune, interacțiune cu mediul, funcție de transfer, lanț de comunicare, canal de comunicare, flux de informație, achiziție, acționare, simptom, diagnoză, reacție, protecție, metamodel, configurație componentă, componentă de platformă, generare automată a configurațiilor.

Utilizând metodologia de cercetare menționată, a fost creată și demonstrată o nouă metodă de proiectare automatizată a sistemelor electronice distribuite.

Noutatea științifică: A fost definit un concept generic de arhitectură a unui sistem încorporat. S-au evidențiat modurile de interacțiune cu mediul și similaritatea între condiționarea semnalelor în fluxurile de intrare cu cele de ieșire. A fost elaborat un concept de *lanț de comunicare*

comun pentru interacțiunile senzor-actuator și comunicarea cu alte dispozitive. Rezultatele cercetării permit elaborarea unei platforme de modelare a sistemelor electronice distribuite în baza modelelor *arhitecturale în straturi* și *lanțuri de comunicare*.

Problema științifică soluționată constă în dezvoltarea de sisteme electronice distribuite în baza modelării de sistem, produselor program și metodelor moderne de descriere a sistemelor prin metamodele pentru arhitecturi în straturi și gestionare a fluxurilor de informații cu lanțuri de comunicare, fapt care a condus la automatizarea procesului de elaborare, configurare și generare de cod sursă de platformă, ceea ce a permis optimizarea timpului alocat modelării de sisteme și funcționalităților esențiale a acestora.

Semnificația teoretică: În urma analizei situației în domeniul modelării sistemelor electronice distribuite prin prisma sistemelor de tip IoT, dar și a proiectării sistemelor încorporate s-a identificat o similitudine în rezolvarea problemelor clasice. Această observație se referă în special la problemele de tip: interacțiune cu mediul extern; condiționare de semnal; identificare de simptome și stabilirea de diagnoze; abordări de protecție, atât a mediului, cât și a sistemului; transferul sigur al informației;

Aceste cercetări au condus la elaborarea unei metodologii de modelare a sistemelor electronice distribuite cu generarea automată a configurației, în cadrul căreia problemele sunt generalizate și prezentate ca componente predefinite care sunt utilizate în proiectare de aplicații la un înalt nivel de abstracție.

În baza modelelor conceptuale de *arhitectură în straturi* și *lanțuri de comunicare* a fost elaborată o metodologie de proiectare a platformelor de resurse programabile pentru sisteme electronice distribuite.

Acest concept de modelare a condus la dezvoltarea soluțiilor inovative pentru proiectarea de sisteme electronice distribuite în vederea abstractizării mediului prin care este transmis semnalul între funcțiile de transfer ale sistemului.

Valoarea aplicativă a lucrării. Metodologia propusă în teză presupune că toate dispozitivele electronice interconectate sunt abstractizate printr-o arhitectură generică. Întregul sistem distribuit poate fi tratat ca un singur dispozitiv electronic cu componente de interacțiune cu mediul extern de tip senzor și actuator. Componentelor de comunicare în acest dispozitiv le revine rolul de interacțiune între subsisteme. Conceptul propus permite realizarea sistemelor electronice distribuite, făcându-se abstracție de localizarea fizică a echipamentelor care fac parte din sistem sau de unitatea de procesare – în cazul aplicațiilor software. Acest fapt se datorează conceptului de *lanț de comunicare*, care permite componentelor aplicației să acceseze resursele sistemului, indiferent de localizarea acestora, considerându-le ca și locale, accesate prin interfețe sau servicii

de sistem. Mecanismele de interacțiune cu mediul extern în metodologia propusă devin subiect de configurare a componentelor cu *arhitectura în straturi* pentru organizarea *lanțului de comunicare*. În cadrul *lanțului de comunicare* are loc gestionarea informației și condiționarea semnalelor, *acest lanț de comunicare* fiind generat automat din definiții de metamodel. În rezultatul unei asemenea generări automate de soluții se obțin resurse software pentru echipamentele hardware incluse în sistem – sistem electronic distribuit, dar și recomandări pentru proiectarea dispozitivelor hardware. În urma aplicării metodologiei propuse se reduce efortul de proiectare pentru interacțiuni cu mediul, efortul principal fiind concentrat pe dezvoltarea aplicației la nivel de sistem.

Implementarea rezultatelor științifice. Metodologia propusă a fost utilizată în elaborarea diverselor sisteme electronice distribuite, printre care:

Sistem de monitorizare a mediului înconjurător prin intermediul amplasării dispozitivelor electronice în diferite puncte carteziene de interes dotate cu senzori în diverși parametri fizici, cum ar fi temperatură, umiditate, luminozitate, CO, mișcare, nivel zgomot. Sistemul permite o construire a hărților de mediu în diverși parametri fizici în baza datelor colectate din rețeaua sistemului electronic distribuit. Prototipul a fost dezvoltat în cadrul companiei Arobs Software SRL, Republica Moldova și prezentat la expoziția Embedded World 2019 – Nuernberg, Germania;

Instalație de uscare pentru fructe și legume pe bază de pelete cu capacitatea de 1,5 tone materie primă pentru compania „Viomarix-Plus" SRL, Republica Moldova. În rezultatul implementării, a fost instalat un sistem de automatizare la instalația de uscare sus-numită, care a contribuit la reducerea consumului de energie cu 15%, constituind 51kW/h;

Sistem de control al brațelor robotice cu 6 grade de libertate, care permite interacțiunea între mai multe brațe pentru a executa manipulări mecanice complexe;

Sistem de iluminare a interiorului clădirii cu lumina ambiantă prin intermediul a patru oglinzi situate la colțurile unei clădiri, pentru urmărirea mișcării soarelui și reflectarea luminii către o țintă specifică, situată în centru. Sistemul a fost proiectat pentru Pavilionul Republicii Moldova la Expoziția de la Milano, 2015, A.O. Asociația Artelor Alternative „ARTWATT";

Sistem de control a unui frigider inteligent cu funcționalități de control temperatură în camera frigorifică și protecție motor pompă de căldură, dotat cu sistem multimedia pentru monitorizarea produselor, recomandări de rețete. Acest prototip a fost elaborat pentru compania Micrologic Design Automation SRL, Chișinău, Republica Moldova.

În scop educațional, a fost elaborată o placă de dezvoltare urmând arhitectura generică a unui sistem încorporat, cu posibilități de experimentare pe partea de senzori, actuatori, interacțiune cu utilizatorul și comunicare. Această placă de dezvoltare este utilizată în procesul de studiu în cadrul lucrărilor de laborator pentru cursurile Microprocesoare și Interfețe și Sisteme Electronice

Incorporate pentru programele de studiu Microelectronică și Nano tehnologii și Inginerie Biomedicală din cadrul Universității Tehnice a Moldovei.

Echipament educațional a fost elaborat și pentru comerț pe piața din Franța, pentru compania Absa-NT. Au fost elaborate două echipamente – ”PIC Evaluation Board” și ”PIC32 Communication Board” dedicat interfețelor de comunicare WiFi, Bluetooth, ZigBee, Ethernet, USB. Pentru aceeași companie a fost proiectată și o platforma robotică mobilă educațională – Road Runner, dotată cu modul GPS și doua procesoare – ARM și Raspberry Pi, ce permite experimentare cu sisteme electronice distribuite. Proiectele au fost livrate cu suporturi software, elaborate conform metodologiei expuse în teză.

Rezultatele cercetărilor au fost integrate în procesul de studiu în cadrul facultății *Calculatoare*, *Informatică și Microelectronică*, pentru programele de studiu *Sisteme Informaționale*, *Tehnologii Informaționale* – cursul *Internetul Lucrurilor*; pentru programele de studiu *Microelectronica și Nano tehnologii*, *Inginerie Biomedicală* și *Inginerie Software* – cursul *Sisteme Încorporate*; pentru programul de studiu *Robotică și Mecatronica* – cursul *Aplicații ale Sistemelor Robotice*. Cursul *Internetul Lucrurilor* a fost nominalizat câștigător, locul I, în cadrul cursurilor digitale pe platforma educațională ELSE a Universității Tehnice a Moldovei.

Rezultatele științifice înaintate spre susținere:

- Metodologie de modelare a sistemelor electronice distribuite bazate pe o arhitectură generică cu componente în straturi.
- Similitudine în realizarea modelelor de gestionare a fluxurilor de achiziții și de acționare.
- Asocierea mecanismelor de diagnoze și protecție cu fluxurile de achiziție și acționare.
- Combinarea modelelor de interacțiune cu mediul extern – achiziție și acționare, cu modelele de interacțiune între dispozitive – comunicare și definirea unui concept unic de gestionare a fluxurilor de informații pentru toate tipurile de interacțiuni.
- Modelare de sisteme electronice distribuite prin definirea de lanțuri de comunicare comun pentru gestionarea fluxurilor de informație.
- Conceptul de proiectare a sistemelor electronice distribuite prin descriere cu metamodele.
- Produs program specializat de configurare și generare automată a codului sursă pentru componentele platformei de proiectare.

Aprobarea rezultatelor cercetărilor. Conceptul, metodele și rezultatele principale expuse în teză au fost publicate în reviste internaționale și naționale, precum și publicate în lucrările conferințelor internaționale, inclusiv:

- Journal of Wseas Transactions on Computer Research. vol. 7, 2019, indexată în SCOPUS, Copernicus-ICI, Semantic Scholar.

- IEEE IEMTRONICS (International IOT, Electronics and Mechatronics Conference), Toronto, Canada, 21 - 24 April 2021, indexată în SCOPUS.
- The 24th IEEE ICSTCC (International Conference on System Theory, Control and Computing), Sinaia, Romania, October 8 - 10, 2020, indexată în Clarivate Analytics Web of Science.
- Journal of Engineering Science. Vol. XXVI (4) 2019. Cat. B+.
- „Akademos", Revistă de știință, inovare, cultură și artă. Nr. 3 (58), 2020. Cat. B.

Publicații la tema tezei. La tema tezei au fost publicate 13 lucrări științifice: trei articole în reviste din bazele de date **Web of Science** și **SCOPUS**, dintre care una ca singur autor; două articole în reviste din Registrul National al revistelor de profil, dintre care una ca singur autor; opt articole în culegeri de lucrări ale conferințelor internaționale; volum total de 5,1 coli de autor.

Structura și volumul lucrării. Teza este compusă din introducere, trei capitole, concluzii generale și recomandări, bibliografie (179 titluri) și 10 anexe. Conținutul de bază al tezei este expus pe 155 de pagini și inserează 118 de figuri și 3 tabele.

Cuvinte-cheie: modelare; sistem electronic distribuit; Internetul lucrurilor (IoT); stivă de componente; arhitectura în straturi; condiționare semnal; lanț de comunicare; generator cod; metamodel; automatizare.

Sumarul capitolelor tezei:

În **Introducere** sunt prezentate actualitatea și importanta temei de cercetare, scopul și obiectivele tezei, este argumentată noutatea științifică și valoarea practică a lucrării.

Capitolul 1, **Analiza situației în domeniul modelării sistemelor electronice distribuite**, reprezintă o trecere în revistă a problemelor în domeniul sistemelor electronice distribuite prin prisma sistemelor de tip IoT și evoluția aplicațiilor în domeniul sistemelor electronice distribuite. Se identifică cerințele privind comunicarea între componentele unui sistem electronic distribuit și se analizează soluțiile contemporane de interacțiune între componentele acestuia. Se face o introducere în abordările actuale în modelarea sistemelor electronice distribuite prin prisma modelării arhitecturale și comunicare între componentele acesteia, dar și a proiectării bazate pe modelare. Se discută despre importanța automatizării în procesul de proiectare a sistemelor electronice distribuite.

În Capitolul 2, **Metode de modelare a sistemelor electronice distribuite**, sunt expuse considerentele de *modelare arhitecturală* pentru sisteme tip IoT în calitate de sisteme electronice distribuite. Se propune un concept generic de arhitectură a unui sistem electronic incorporat cu definirea componentelor cu modele *arhitecturale în straturi*. Se accentuează importanța componentelor de tip senzor-actuator și se propune un concept de modelare a acestor componente

prin definirea fluxurilor de informație, pe care le gestionează aceste componente. Se asociază funcționalitățile de diagnoză cu fluxul de achiziție a datelor și funcționalitățile de protecție cu fluxul de acționare. În ceea ce privește analiza fluxurilor de informație, se propune o metodă de modelare a sistemelor electronice distribuite bazată pe *lanțuri de comunicare* formate din componente specializate și se propune un instrument-platformă de modelare cu metamodele. Componentele de nivelul aplicației sunt calificate ca sursa și destinație a *lanțurilor de comunicare*. Totodată pentru componentele aplicațiilor distribuite pe sisteme electronice distribuite, *lanțurile de comunicare* propuse realizează și abstractizează interacțiunile între componentele.

În Capitolul 3, **Modelarea arhitecturală a sistemelor electronice distribuite. Studii de caz**, este prezentat un instrument de modelare prin metamodele și generare automată de cod sursa pentru componentele de platformă, produs program specializat realizat ca resursă software pentru punerea în aplicație practică și validare a conceptului propus în teză. Sunt expuse aplicațiile practice ale metodei de modelare prin lanțuri de comunicare, evidențiindu-se diversitatea domeniilor de aplicare, cum ar fi sistem de monitorizare a parametrilor mediului, sisteme de control a echipamentelor de mecatronică, sistem de control a procesului de producere, dispozitive electronice electrocasnice.

1. ANALIZA SITUAŢIEI ÎN DOMENIUL MODELĂRII SISTEMELOR ELECTRONICE DISTRIBUITE

1.1 Sisteme electronice distribuite prin prisma sistemelor tip IoT

1.1.1 Evoluția aplicațiilor în domeniul sistemelor electronice distribuite

Internetul Lucrurilor sau Internet of Things (IoT) reprezintă o rețea sau un sistem de obiecte sau lucruri fizice, cum ar fi – dispozitive, clădiri, automobile, obiecte de uz casnic și alte unități electrice și mecanice, sisteme biologice, chimice și optice încorporate cu electronică, software, senzori și conexiunea la rețea, permițând acestor dispozitive să comunice între ele și să facă schimb de date. Astfel, acest sistem ne permite să monitorizăm starea acestor dispozitive, să le controlăm de la distanță, să le programăm comportamentul și să obținem mesaje de notificare pentru anumite evenimente de interes.

Odată cu începutul secolului XXI, IoT, a început să se dezvolte odată cu convergența diferitelor tehnologii, începând cu comunicații fără fir (wireless) prin Internet, de ex. WiFi (802.11), sisteme Embedded performante la cost mic, cum ar fi diverse microcontrolere (MCU) pe arhitectura AVR, ARM, PIC ș.a. Avansarea substanțială din domeniul microelectronicii, tehnologiilor VLSI și ULSI a permis proiectarea diferitor dispozitive și sisteme micro-electro-mecanice (MEMS) și procesoarelor cu matrice de porți logice programabile (Field Programmable Gate Array – FPGA) specializate în procesare și imagistică. Dezvoltarea produselor și programelor pentru proiectarea asistată de calculator (Computer Aided Design – CAD) a contribuit la posibilitatea de a integra diverse dispozitive pe scară largă de producție și cost mic pentru a crea sisteme complexe de monitorizare, control și automatizare.

Conceptul unei rețele de dispozitive electronice inteligente a apărut încă în 1982, la Universitatea Carnegie Mellon, unde o mașină de vânzare Coca-Cola a devenit primul dispozitiv de uz general conectat la Internet, care putea să raporteze când s-au terminat sticlele de vânzare și daca s-a răcit băutură [1]. În 1991 Mark Weiser în lucrarea sa „The Computer of the 21st Century" [2] a discutat o viziune modernă asupra conceptului de IoT, dezvoltat în lucrarea "From The Internet of Computers to the Internet of Things" de F. Mattern și C. Floerkeimer [3].

În 1994, Reza Raji a descris acest concept ca transmiterea pachetelor mici de date unei mulțimi mari de noduri a unei rețele, pentru a integra și automatiza cât se poate de multe lucruri, începând cu lucruri de uz casnic până la uzine întregi [4]. Conceptul de IoT a devenit popular în 1999 prin intermediului Auto-ID Center la MIT și diverse publicații mass-media, unde a fost propusă și popularizată identificarea obiectelor cu ajutorul radio-frecvențelor (Radio Frequency Identification – RFID). Astfel, la orice obiect poate fi anexat un tag RFID, ce reprezintă practic o

antenă cu microprocesor și puțină memorie pentru stocarea codului unic de identificare, astfel orice produs alimentar, dispozitiv, haina ș.a, pot fi identificate unic cu ajutorul unui sistem emițător-receptor de frecvență radio. Conform raportului Organisation for Economic Co-operation and Development (OECD), în 2015, numărul de dispozitive IoT online este în creștere continuă, astfel pe primul loc se află Coreea de Sud cu circa 37.9 dispozitive la 100 de oameni.

O prognoză recentă făcută de International Data Corporation (IDC) estimează IoT și ecosistemul asociat să fie o piață de 1,7 trilioane de dolari până în 2020, care va include 212 miliarde de lucruri conectate. IoT va alimenta o schimbare de paradigmă a unei lumi „cu adevărat conectate", în care obiectele de zi cu zi devin interconectate și inteligente cu capacitatea de a comunica mai multe tipuri diferite de informații între ele, precum și cu utilizatorii umani. Fig. 1.1, prezintă un grafic de evoluție a exploziei IoT pentru perioada anilor 1992-2020, conform rapoartelor Cisco [5].

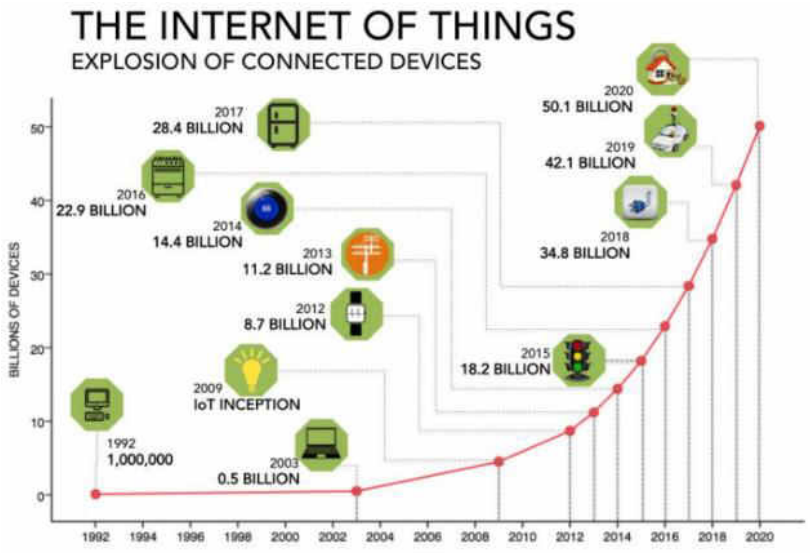


Fig. 1.1. Explozia IoT în perioada 1992-2020 [5]

IoT este o simbioză a diverselor tehnologii și va schimba drastic ceea ce se poate realiza de pe Internet în prezent. IoT funcționează cu diverse tehnologii abilitate și emergente, cum ar fi rețelele de senzori fără fir (Wireless Sensor Network – WSN) [6], tehnologiile senzorilor, învățarea automată și inteligența artificială (AI), big data și analiza etc. În centrul Institute of Technology Tralee (Ireland) ITT sunt instalate WSN-uri, constând din senzori implementați într-o zonă de detectare pentru a monitoriza fenomene specifice cum ar fi monitorizarea mediului și colectarea datelor [7]. Se dezvoltă o configurație de rețea mult mai distribuită, în cazul în care toate dispozitivele posibile, în mare parte de natură eterogenă, se conectează unul cu altul pentru a achiziționa și a analiza date de natură diferită, pentru a acționa pe baza inteligenței obținute din cunoștințele profunde ale datelor. Aceste acțiuni sunt în mare parte fără interacțiune umană [8].

Cuvântul IoT a fost popularizat pentru prima dată de Kevin Ashton în 1999, când RFID a fost utilizat pentru aplicarea în gestionarea lanțului de aprovizionare [9]. De atunci, IoT a fost folosit pentru a defini o paradigmă a oricăror dispozitive posibile, care pot fi conectate la Internet pentru colectarea datelor, formarea cunoștințelor și automatizarea. Conceptul de IoT a generat multă atenție din partea guvernelor, industriilor și cercetătorilor.

Trăim în era modernă, în care timpul și banii devin tot mai critici în toate domeniile de activitate. Aceasta include și domeniul dezvoltării de software, în special domeniul auto, deoarece OEM-urile (Original equipment manufacturer – OEM) doresc să adauge mai multe funcționalități în platformele existente sau platforme noi care să includă aceste funcționalități, dar realizate cu aceleași constrângeri financiare și temporale. O mai mare funcționalitate implică o complexitate tot mai mare a software-ului dezvoltat, un număr mare de variante pentru aceeași platformă auto prin aceleași sau separate unități de control electronic (ECU) [10].

Progresul rapid al tehnologiilor hardware și de comunicații din ultimele decenii a permis dezvoltarea unei game largi de aplicații de conectivitate. Serviciile de comunicații în bandă largă și cloud îmbunătățite permit conectarea mai multor dispozitive la rețeaua Internet, fenomen cunoscut în general ca „Internetul lucrurilor” sau IoT. Cu toate acestea, caracterul perturbator al IoT necesită evaluarea cerințelor pentru implementarea sa viitoare în lanțul valoric digital în diferite industrii și în multe domenii de aplicație [11]. Internetul Industrial al lucrurilor (Intenrnet Industrial of Things – IIoT) introduce prin Internet dispozitive permise în sistemele de proces industrial, care operează în sectoare precum energetică, transport, asistență medicală, utilități, orașe, agricultură, infrastructură etc. IIoT poate fi privit ca un sistem de sisteme. Arhitectura unui singur sistem IIoT constă din diferite straturi. Fiecare strat îndeplinește o funcție distinctă, cu caracteristici operaționale unice, și se bazează pe dispozitive și protocoale de comunicații diferite decât alte straturi ale sistemului [12]. Ulterior, companiile investesc eforturi semnificative în dezvoltarea tehnologiilor din Industria 4.0, pentru a-și îmbunătăți operațiunile actuale și a oferi o propunere de valoare mai competitivă clienților existenți și noilor clienți [13]. IoT permite dispozitivelor de zi cu zi să devină mai inteligente, să proceseze mai inteligent și să dezvăluie comunicarea. În timp ce IoT își caută încă platforma, prin efectele sale acesta este privit deja ca un instrument de soluție universală pentru sistemele de echipamente distribuite [14].

În ultima sa dezvoltare, IoT a început să fie denumit Internetul a orice, deoarece orice obiect din lume este un potențial participant la rețeaua mondială. Deoarece numărul lucrurilor conectate la Internet crește exponențial, reutilizarea soluțiilor existente este un punct semnificativ pentru a face față cererii mari pentru noul tip de lucruri care trebuie conectate la rețea. Pe lângă reutilizarea în sine, abordarea arhitecturală corectă care permite reutilizarea este conceptul cheie,

în care IoT este văzut ca un sistem întreg, continuu, în ceea ce privește un dispozitiv distribuit care simte totul și acționează asupra a tot ceea ce este în lume. Din stratul arhitectural inferior, sistemul poate fi privit ca un Internet al unităților senzor-actuator. În consecință, aplicațiile ar putea fi dezvoltate pentru a funcționa cu fiecare senzor și pentru a acționa prin fiecare actuator din rețea. În plus, din acest concept, componentele cheie sunt componentele modelului senzor-actuator și conducerea semnalelor către aplicații.

IoT are diverse aplicații și oferă oportunități enorme de dezvoltare și implementare continuă a acestor sisteme. Astfel Gartner Inc. prevedea că vor fi aproximativ 26 miliarde de dispozitive conectate la Internet până în 2020 [15], iar ABI Research ridica această cifă până la 30 de miliarde [16], o cifră ce depășește considerabil populația globului. Aceste dispozitive integrate cu resurse de procesare și memorie limitate au aplicabilitate practică în foarte multe sisteme, începând cu achiziția datelor din ecosisteme naturale și terminând cu sisteme de case inteligente, uzine, stații nucleare ș.a. Astfel de sisteme nu se limitează doar la colectarea datelor și monitorizarea diferitor procese.

Printre exemplele cu un potențial enorm al sistemelor IoT pot fi luate în considerare:

- Sisteme inteligente de cumpărături – pot monitoriza articolele achiziționate pentru diferiți utilizatori analizând profilul social și construind sisteme de recomandare pentru a oferi oferte viitoare personalizate, pe baza unor criterii precum locul în care anumite articole pot fi achiziționate, cel mai rezonabil preț, cea mai bună calitate;
- Frigidere inteligente – încorporare de dispozitive inteligente și conexiunea la rețea a unui frigider. Unele soluții sunt deja existente, cum ar fi Samsung [17]. Frigiderul inteligent permite utilizatorilor să fie mereu la curent cu produsele disponibile;
- Sisteme de achiziție de date de mediu pe suprafețe mari – au o vastă zonă de utilizare pentru a monitoriza diferiți parametri precum temperatura, umiditatea, concentrația diferitelor tipuri de gaz, vibrațiile din scoarța terestră, viteza vântului ș.a. Acest lucru permite efectuarea de analize statistice, prognoze meteo, evaluare calitativă a ecosistemului, toate direct online pe o pagină web sau pe orice telefon inteligent, accesibile în toată lumea (cu acces la Internet);
- Coordonarea activităților întreprinderilor sistemele IoT pot fi utilizate pentru managementul întreprinderii pentru a identifica când angajații intră sau ies din birou și pentru a calcula timpul lor total de lucru sau la o scară mai mare, monitorizând activitatea persoanelor pentru a gestiona eficient resursele umane. O altă aplicație a acestei metode poate fi reglarea ambuteiajelor auto. De exemplu, crearea unei rețele de dispozitive pentru identificarea automobilelor și a fluxului în trafic poate fi utilizată pentru a calcula și redirecționa fluxul în timp real folosind diferite modele matematice. Acest lucru permite să minimizăm congestia și să creștem eficiența;

- Sisteme medicale – dispozitivele IoT pot fi integrate în diverse dispozitive medicale. De exemplu, ar putea servi un sistem încorporat de senzori biologici care monitorizează tensiunea arterială, prin crearea diagramei ECG în timp, prin obținerea de date despre biochimie unică, stare hormonală și multe altele;
- Casa Inteligentă – încorporarea dispozitivelor inteligente în casă și dispozitive de uz casnic, cu diverse aplicații la îndemână, cum ar fi monitorizare video, monitorizarea energiei totale și individuale consumate a fiecărui dispozitiv, achiziționarea datelor despre temperatura camerei, concentrația diverselor tipuri de gaz în aer, sisteme de siguranță și alarmă la incendiu, conectarea și deconectarea dispozitivelor la distanță

Se poate observa că conceptul de IoT aduce multe oportunități pentru dezvoltarea aplicațiilor. Unele dintre soluții nu sunt încă implementate în practică. Acest fapt oferă tehnologiei de astăzi un potențial enorm și se așteaptă să apară o mulțime de soluții și implementări. Acest lucru va duce la un impact extraordinar asupra tuturor aspectelor vieții umane. În special, împreună cu acceptarea lor largă, acest tip de sistem va deveni din ce în ce mai accesibil. Dezvoltarea în timp a diferitelor aplicații IoT este reprezentată în Fig. 1.2

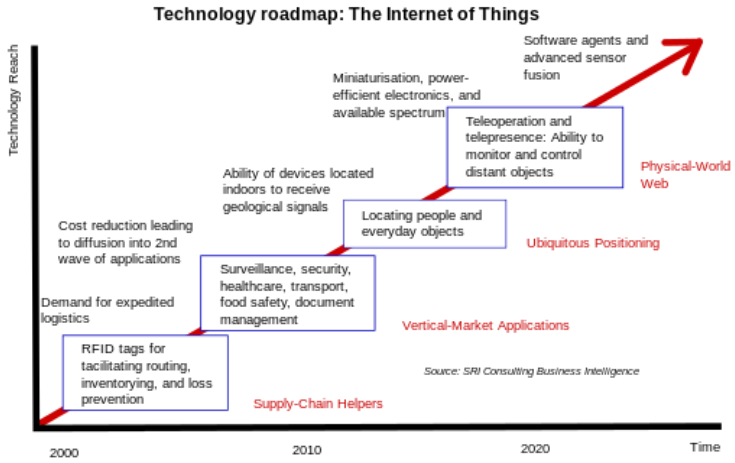


Fig. 1.2. Dezvoltarea tehnologiilor IoT în timp [18]

Fără aplicații, IoT nu are sens. Aplicațiile IoT asigură livrarea mesajelor în timp real și comunicații fiabile. Acestea introduc toate funcționalitățile sistemului utilizatorului final printr-o multitudine de dispozitive conectate. Conectivitatea fizică este realizată de rețele și dispozitive, în timp ce interacțiunile robuste dintre dispozitiv-dispozitiv și om-dispozitiv sunt furnizate de aplicațiile IoT [19]. În aplicațiile pentru dispozitive umane, vizualizarea este considerată una dintre caracteristicile cheie care permit utilizatorului să interacționeze cu ușurință cu mediul, să prezinte și să înțeleagă în mod eficient informațiile colectate [20, 19]. În aplicațiile de la dispozitiv la dispozitiv, informația este de obicei implementată pentru a permite interacțiuni dinamice. Acest

lucru permite dispozitivelor să monitorizeze automat mediul, să identifice problemele, să colaboreze și să ia în mod independent deciziile adecvate fără intervenția umană [19].

Cerințe privind comunicarea între componentele sistemului

În funcție de aplicație, poate exista o gamă largă de cerințe pentru transmiterea datelor. Acest lucru poate fi realizat prin mai multe protocoale de comunicare bazate pe scopuri și condițiile subiectului. Unele exemple de cerințe pentru comunicare sunt prezentate în Tabelul 1.1. După cum se poate vedea, în toate aplicațiile, securitatea reprezintă sarcina esențială. Cu toate acestea, multe aplicații au nevoi diferite în ceea ce privește latența sau viteza de transmisie necesare, astfel încât sistemele IoT trebuie să fie compatibile. Indiferent de tehnologiile și metodele aplicate într-un sistem, fiecare aplicație ar trebui să poată comunica eficient tuturor celorlalte componente.

În multe cazuri, securitatea transmisiei nu este crucială, de exemplu, transmisia audio și video ar trebui să aibă o latență scăzută și o viteză mare de transmisie, dar dacă unele informații se pierd pe drum, acest lucru nu este esențial și de aceea sistemele IoT acceptă diferite protocoale, inclusiv TCP / IP, UDP. De exemplu, folosind protocolul UDP, nu există retransmisie a mesajelor pentru a detecta erorile de transmisie, deci într-un mediu cu zgomot ridicat, mesajele nu vor fi transmise de multe ori, astfel încât banda va fi utilizată foarte eficient, ceea ce nu este valabil pentru TCP în acest caz. Există și protocoale de nivel superior, cum ar fi SSH (Secured SHell) sau HTTPS (HyperText Transfer Protocol Secure), care pot fi utilizate pentru a asigura securitatea transmiterii datelor prin criptare.

Tabelul 1.1 Cerințe față de comunicare pentru diverse aplicații

	Latență	Viteza de transmitere	Securitatea	Consumul de energie	Siguranța transmiterii
Sisteme biomedicale	Medie	Mare	Mare	Medie	Mare
Automotive	Mică	Medie	Mare	Medie	Mare
Transmitere audio	Mică	Mare	Mare	Medie	Medie
Transmitere video	Mică	Mare	Mare	Medie	Mică
Dispozitive de uz casnic	Mare	Mică	Mare	Medie/Mare	Medie

Componentele esențiale ale unui sistem IoT pentru funcționarea deplină a dispozitivelor de colectare și control prin Internet (Fig. 1.3) sunt următoarele:

- Obiectul – entitatea fizică care urmează să fie controlată și să adune date. Este, de obicei, un dispozitiv care are o interfață de comunicație electronică și metode de interacțiune cu mediul, asupra căruia pot fi exercitate diferite mecanisme de control;

- Microcontroler – un microprocesor cu cipuri integrate utilizat pentru a realiza programul de control, împreună cu colectarea, procesarea și transmiterea datelor către alte componente;
- Gateway – un dispozitiv sofisticat, al cărui scop este să colecteze date de la mai multe obiecte (prin microcontrolere), să efectueze o agregare logică, să le proceseze și să le transmită pentru stocare și procesare către rețeaua Internet;
- Internet – rețeaua globală de transmitere a datelor, care, din perspectiva IoT, ne permite să monitorizăm și să controlăm dispozitivele de la distanță;
- Client – interfața prin care utilizatorul cu acces la Internet poate comunica cu dispozitivele, fie de pe un computer, telefon inteligent ș.a.

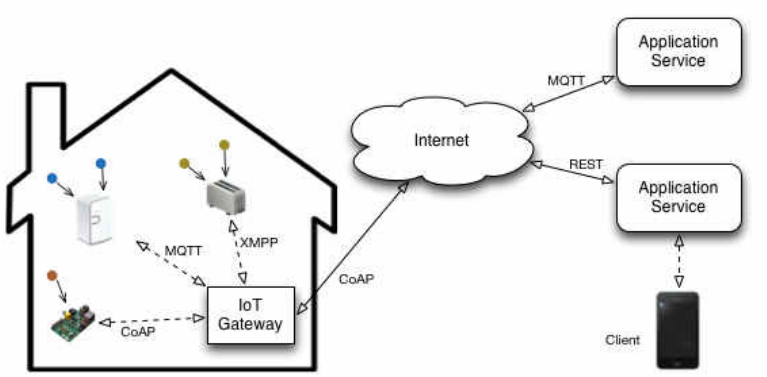


Fig. 1.3. Principalele componente ale unui sistem IoT [21]

Sistemele IoT sunt supuse la două constrângeri principale pentru a fi fezabile – costul și performanța. Acest lucru induce o gamă largă de cerințe și principii cu privire la sistemele IoT, care vizează proiectarea și organizarea acestora, după cum urmează:

- Identitate – fiecare dispozitiv trebuie identificat în mod unic. Această presupunere este banală pentru funcționarea corectă a acestor tipuri de sisteme;
- Securitate – dispozitivele trebuie să transmită în siguranță date, utilizând mecanisme de autorizare. Securitatea este extrem de esențială pentru multe aspecte ale sistemului: începând de la funcționarea sistemului până la confidențialitatea datelor utilizatorilor și drepturile de protecție pentru siguranța și chiar viața utilizatorilor;
- Independență și interoperabilitate – dispozitivele trebuie să fie independente. Dacă un dispozitiv nu funcționează, acesta nu trebuie să aibă o influență decisivă asupra funcționării unui alt dispozitiv. Dispozitivele trebuie să comunice printr-un protocol bine definit și standardizat, pentru a asigura schimbul și interoperabilitatea dispozitivelor;
- Consum redus de energie – acesta este un criteriu esențial pentru dispozitivele mici de achiziție de date. Consumul redus de energie va asigura o durată lungă de viață a bateriei și, respectiv, eficiența costurilor.

Un sistem este scalabil dacă este posibilă adăugarea de noi servicii și echipamente fără a-i degrada performanța. Problema principală cu IoT este operarea cu un număr mare de dispozitive cu memorie, procesare, spațiu de stocare și bandă de transfer diferite [22]. O altă problemă importantă care trebuie luată în considerare este disponibilitatea. Scalabilitatea și disponibilitatea necesită să fie implementate împreună în cadrul sistemelor IoT. Un exemplu excelent de scalabilitate îl reprezintă sistemele IoT bazate pe cloud, care oferă suport suficient pentru a scala rețelele IoT prin adăugarea de noi dispozitive, stocare și putere de procesare, după necesitate.

Cu toate acestea, rețeaua globală IoT distribuită creează o nouă paradigmă de cercetare pentru a dezvolta un cadru IoT, care să satisfacă nevoile globale [23]. O altă provocare cheie este disponibilitatea resurselor pentru obiectele autentice, indiferent de locația lor și de momentul cererii. În mod distribuit, mai multe rețele IoT mici conectate la platformele globale IoT, pentru pune la dispoziție resursele și serviciile sale, dar și pentru a le utiliza pe cele din platforme. Prin urmare, disponibilitatea este o preocupare importantă [24]. Datorită utilizării diferitelor canale de transmisie a datelor, cum ar fi comunicații prin satelit, unele servicii și disponibilitatea resurselor pot fi întrerupte. Prin urmare, este necesar un canal de transmisie de date independent și fiabil pentru disponibilitatea neîntreruptă a resurselor și serviciilor [25].

1.1.2 Soluții contemporane de interacțiune în sisteme electronice distribuite

Indiferent care ar fi scopul unei aplicației cu dispozitive electronice distribuite, este cert faptul că acestea au nevoie de o metodă de a se interconecta, în scopul de a face schimb de informații și utilizare servicii. Comunicarea în aplicațiile IoT cuprinde în mod normal următoarele tipuri de interacțiuni [26]:

- Conexiune People to People (P2P) –transferul de date de la o persoană la alta. Apare prin apel video, apel telefonic și comunicări sociale;
- Conexiune Machine to People (M2P) – transferul de informație între dispozitive și utilizatorul uman. De exemplu, prognoza meteo folosește dispozitive inteligente pentru a aduna datele din mediu și a le trimite înapoi administratorilor din centrul de control pentru analize ulterioare;
- Conexiune Machine to Machine (M2M) – transferul de date între dispozitive, fără interacțiuni umane. De exemplu, o mașină care comunică cu o altă mașină despre valoarea vitezei, schimbarea benzii sau intenții de frânare etc.

Dispozitivele IoT se conectează, de obicei, la Internet prin stiva IP (Internet Protocol). Această stivă este foarte complexă și necesită o cantitate mare de energie și memorie de la dispozitivele de conectare. Dispozitivele IoT se pot conecta și local – prin rețele non-IP, care consumă mai puțină energie și se pot conecta la Internet printr-un gateway inteligent. Canalele de

comunicare non-IP, precum Bluetooth, RFID și NFC, sunt destul de populare, dar limitate în raza lor de acțiune (până la câțiva metri). Prin urmare, aplicațiile lor sunt limitate la rețele personale mici. Rețelele personale (PAN) sunt utilizate pe scară largă în aplicațiile IoT, cum ar fi dispozitivele portabile conectate la telefoane inteligente. Pentru creșterea gamei de astfel de rețele locale, a fost necesară modificarea stivei IP, astfel încât să faciliteze comunicarea cu un consum redus de energie utilizând stiva IP. Una dintre soluții este 6LoWPAN, care încorporează IPv6 cu rețele de zonă personală de mică putere. Gama unui PAN cu 6LoWPAN este similară cu rețelele locale, iar consumul de energie este mult mai mic [27].

Cele mai populare tehnologii de comunicare utilizate în domeniul IoT sunt IEEE 802.15.4, WiFi de consum redus, 6LoWPAN, RFID, NFC, Sigfox, LoRaWAN și alte protocoale specializate pentru rețelele fără fir.

Pentru IoT în calitate de sisteme electronice distribuite, interacțiunile sunt realizate la nivel de conexiuni Machine to Machine (M2M). Acestea sunt realizate prin diverse tehnologii cu fir (wired) și fără fir (wireless) la nivel de hardware și software.

Tehnologiile fără fir pot fi clasificate, în funcție de gama de frecvențe, în patru domenii, Fig. 1.4. Primul domeniu este gama de proximitate mică și au o rază de acțiune de la 0 la 10 metri, cum ar fi tehnologia RFID [28]. Al doilea domeniu este gama scurtă cunoscută sub numele de rețea de zonă personală fără fir (WPAN), cu o gamă de la 10 la 100 de metri, cum ar fi tehnologiile Bluetooth și ZigBee. Al treilea domeniu este intervalul scurt / mediu, care a fost identificat ca rețea locală fără fir (WLAN), cu o rază de la 100 până la 1000 de metri, cum ar fi rețelele Wi-Fi [28, 29]. În cele din urmă, distanța lungă, cunoscută ca rețeaua wireless extinsă (WWAN) cu o autonomie de până la 100 km, cum ar fi celulară și LPWAN. Tehnologia LPWAN este una dintre cele mai populare tehnici utilizate în rețelele WWAN. Rețeaua de tip LoRaWAN, care se bazează pe tehnologia LoRa, este cea mai populară rețea de putere redusă cu o rază lungă de acțiune, fiind convenabilă pentru dezvoltarea sistemelor IoT [30].

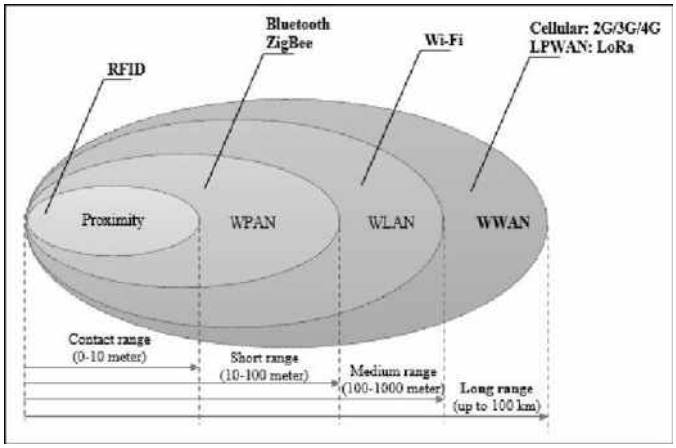


Fig. 1.4. Clasificarea tehnologiilor fără fir pe baza domeniului [31]

Soluții cu rețele LoRaWAN

Majoritatea rețelelor LPWAN folosesc topologia rețelei stea. Natura sa este identificată pe baza unui nod central, acționează ca gateway și se conectează cu toate celelalte dispozitive conectate [28]. Pe de altă parte, topologia rețelei tip mesh (rețea de tip plasă) este formată din noduri individuale care propagă date către celelalte noduri pentru a crește gama de comunicații. Implementarea rețelei LoRaWAN se bazează pe topologia rețelei tip stea. Avantajele utilizării topologiei tip stea sunt păstrarea duratei de viață a bateriei și scăderea complexității rețelei. Nodurile nu trebuie să propage sau să transmită alte date către alte noduri, nodurile primind doar datele proprii. Fig. 1.5 ilustrează arhitectura LoRaWAN, care poate fi împărțită în părți front-end și back-end [31, 32]. Partea front-end conține elemente gateway și nodurile finale, iar partea back-end conține serverele de rețea responsabile pentru verificarea securității, stocarea informațiilor primite, filtrarea pachetelor duplicate și programarea confirmărilor prin gateway [33].

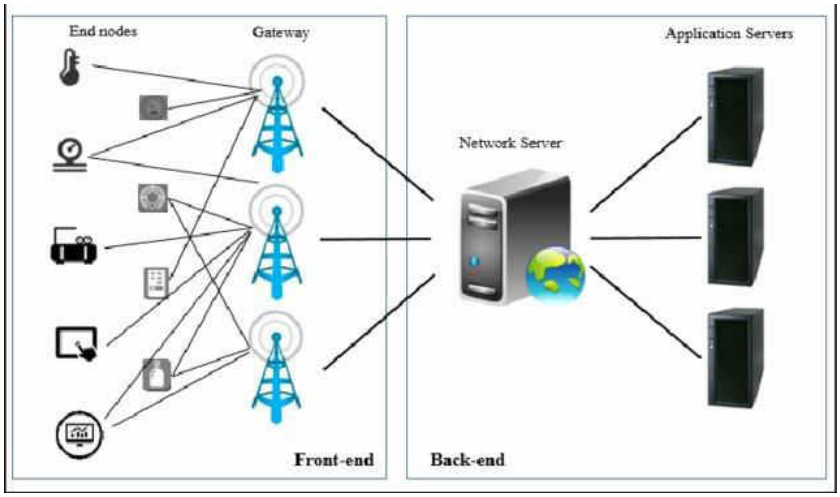


Fig. 1.5. Arhitectura LoRaWAN [32]

Dispozitivele cu nod final LoRaWAN au trei moduri de funcționare, în funcție de instrumentul de comunicare utilizat: Clasa A (implicit), Clasa B și Clasa C (ambele opționale) [34]. Aceste trei moduri descriu felul în care dispozitivul cu nod final poate accesa o rețea fără fir. Clasa A se concentrează pe traficul ascendent dintre dispozitivul de nod final și serverul de rețea, în plus, permite comunicarea bidirecțională. Clasa B și clasa C îmbunătățesc traficul descendent de la serverul de rețea la dispozitivul de nod final prin utilizarea caracteristicilor opționale. Cercetările actuale se concentrează pe îmbunătățirea de performanțe a acestor clase, în special pe economisirea duratei de viață a bateriei și a ratei de livrare a pachetelor [32]. Există multe alte tehnologii LPWAN legate de tehnologiile de comunicare IoT, care au aceeași specificație ca LoRaWAN. Una dintre aceste tehnologii este așa-numitul protocol Sigfox [35], zona de acoperire Sigfox este mai bună decât LoRaWAN. Avantajul LoRaWAN față de Sigfox este în mare parte

bazat pe faptul că este un protocol deschis. Un alt exemplu este tehnologia protocolului cu bandă îngustă (NB-IoT), care are multe caracteristici similare cu LoRaWAN, bandă de înaltă frecvență, dar NB-IoT nu este un protocol deschis, spre deosebire de LoRaWAN [36]. LoRaWAN are alte avantaje care nu sunt incluse în NB-IoT, inclusiv scalabilitatea lățimii de bandă, eficiența sporită a energiei și rata de adaptare a datelor. Mai multe exemple de protocol similare cu LoRaWAN sunt Weightless, NWAVE, OnRamp wireless, IEEE802.11ah și Dash7 [37, 35, 38, 39]. Fig. 1.6 prezintă cele mai populare tehnologii LPWAN [30].

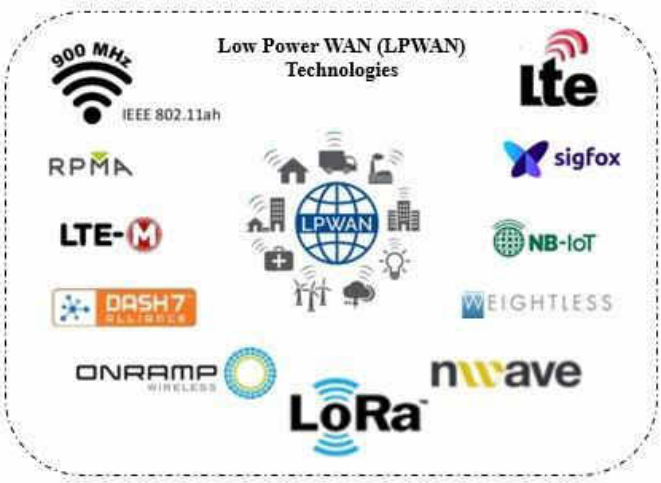


Fig. 1.6. Tehnologii WAN de consum redus (LPWAN) [30]

LoRa este o platformă de tehnologie fără fir, cu rază lungă de acțiune, care utilizează spectru radio fără licență în banda radio industrială, științifică și medicală (banda ISM) [40]. LoRa are ca scop eliminarea repetitoarelor de semnal intermediare, reducerea costurilor dispozitivelor, creșterea duratei de viață a bateriei pe dispozitive, îmbunătățirea capacității rețelei și acceptarea unui număr mare de dispozitive. Este un strat fizic utilizat pentru comunicarea pe distanțe lungi. Pentru a obține o putere redusă, majoritatea tehnologiilor fără fir utilizează modulația cu frecvență Shift Key (FSK). Cu toate acestea, LoRa folosește modulația Chirp-spread-spectrum (CSS) pentru a menține caracteristicile de putere reduse în beneficiul creșterii intervalului de comunicare. LoRa este prima implementare pentru infrastructura low-cost comercializată folosind CSS. CSS a fost utilizat în comunicațiile pe distanțe lungi de către agențiile militare și spațiale datorită capacității sale de a rezista la interferențe. LoRaWAN este un protocol de comunicații fără fir, dezvoltat de LoRa Alliance pentru a servi provocărilor cu care se confruntă comunicarea pe distanțe lungi cu IoT. Rețelele LoRa se referă în mod specific la soluții cu consum redus de energie, cu o rată de date transmise redusă, datorită arhitecturii sale de sistem bazate pe LoRaWAN. Protocolul și arhitectura rețelei LoRaWAN au o mare influență asupra determinării duratei de viață a bateriei, capacității rețelei, calității serviciului (QoS), securității și diversității de aplicații în rețea [41].

Comunicare în câmp aproape (Near Field Communication - NFC)

NFC este o tehnologie de comunicație fără fir cu rază foarte scurtă de acțiune, prin care dispozitivele mobile pot interacționa între ele pe o distanță de doar câțiva centimetri [42, 43, 44]. Toate tipurile de date pot fi transferate între două dispozitive compatibile NFC în câteva secunde, apropiindu-le unul de celălalt. Această tehnologie se bazează pe RFID. Folosește variații ale câmpului magnetic pentru a comunica date între două dispozitive NFC. NFC funcționează pe o bandă de frecvență de 13,56 MHz, care este aceeași cu RFID de înaltă frecvență. Există două moduri de funcționare: activ și pasiv. În modul activ, ambele dispozitive generează câmpuri magnetice, în timp ce în modul pasiv, un singur dispozitiv generează câmpul, iar celălalt folosește modulația sarcinii pentru a transfera datele. Modul pasiv este util în dispozitivele alimentate cu baterii pentru a optimiza consumul de energie. Un avantaj al cerinței de apropiere între dispozitive este că este util pentru tranzacții sigure, cum ar fi plățile. Spre deosebire de RFID, NFC poate fi utilizat pentru comunicarea bidirecțională. În consecință, aproape toate smartphone-urile de pe piață sunt dotate cu module NFC [27].

Rețele de senzori fără fir (Wireless Sensor Network – WSN) bazate pe IP pentru obiecte inteligente

De multe ori, datele de la un singur senzor nu sunt utile în monitorizarea suprafețelor mari și a activităților complexe. Diferite noduri de senzori trebuie să interacționeze fără fir. Dezavantajul tehnologiilor non-IP, precum RFID, NFC și Bluetooth, constă în faptul că gama lor este foarte mică. Prin urmare, acestea nu pot fi utilizate în multe aplicații, unde o suprafață mare trebuie monitorizată prin multiple noduri de senzori desfășurate în locații diverse. O rețea de senzori fără fir (WSN) este formată din zeci până la mii de noduri de senzori conectate, utilizând tehnologii fără fir. Nodurile colectează date despre mediu și le comunică dispozitivelor gateway, care transmit informațiile către cloud prin Internet. Comunicarea dintre noduri dintr-un WSN poate fi directă sau multi-hop. Nodurile senzorilor sunt de natură cu putere de execuție limitată, dar nodurile gateway-ului au suficientă putere și resurse de procesare. Topologiile populare de rețea utilizate într-un WSN sunt rețea de tip stea sau o rețea hibridă. Majoritatea comunicațiilor din WSN se bazează pe standardul IEEE 802.15.4. Există în mod clar o mulțime de protocoale, care pot fi utilizate în scenarii IoT [27].

Rețeaua de senzori fără fir permite senzorilor să comunice fără nicio șansă de a utiliza fire pentru comunicare, precum și pentru transmiterea datelor [45]. Deci, prin comoditatea rețelelor de senzori fără fir, precum și importanța lor în viața de zi cu zi, oamenii de știință au dezvoltat și cercetat numeroase rețele de senzori fără fir [46, 47], prin utilizarea senzorilor. În general, WSN-urile reprezintă un subiect important de studiu, în care mulți cercetători au încercat să beneficieze

de acesta pentru a construi diverse sisteme în care sunt implicate controlul, urmărirea sau monitorizarea, cum ar fi rețeaua inteligentă [48, 49, 50, 51, 52], clădirile inteligente [53, 54, 55, 56], ciclism pe traseu [57, 58, 59], localizare [60, 61], alarmă inteligentă, monitorizare, control și gestionare a energiei [62, 63, 64, 65, 67, 68, 69], îngrijire a sănătății [70], agricultură [71] și multe alte aplicații. Rețeaua de senzori fără fir este o tehnologie formată dintr-un număr mare de noduri și are mai multe aplicații, cum ar fi analiza temperaturii și umidității unui anumit spațiu geografic, supravegherea, sunetul, prezența individuală, în medicină, domeniul militar, în agricultură și, printre altele, în funcție de domeniile lor de utilizare și studii [72]. Cu toate acestea, pentru a putea face schimb de informații între nodurile senzorului sau între nodurile senzorului cu nodul routerului sau între router și coordonator, utilizează tehnologii de comunicații [73, 74, 75], cum ar fi Wi-Fi, Bluetooth , ZigBee și RF. În plus, problema consumului de energie în rețelele de senzori fără fir este de mare interes, în care furnizarea de energie și colectarea energiei din mediul înconjurător este un domeniu nou de cercetare. Aplicarea tehnologiilor de transmisie a energiei fără fir în acest caz este mai importantă ca niciodată [76, 77]. Pe lângă economia de energie în rețelele de senzori fără fir, utilizarea protocoalelor de comunicații este o cerință foarte importantă. Conform [78, 79, 80], aceste protocoale sunt divizate pe categorii, cum ar fi: cele bazate pe modul de funcționare și tipurile de aplicații țintă; în funcție de modul de participare al nodurilor; în funcție de structura rețelei, protocoalele de diseminare a datelor și protocoalele de colectare a datelor [81].

Integrarea RFID și WSN

Rețelele de senzori RFID și wireless (WSN) sunt tehnologii importante în domeniul IoT. RFID poate fi utilizat numai pentru identificarea obiectelor, dar WSN-urile au un scop mult mai mare [27]. Cele două sunt foarte diferite, dar fuzionarea acestora are multiple avantaje. Următoarele componente pot fi adăugate la RFID pentru a-i îmbunătăți gradul de utilizare: capabilități de detectare, comunicare multi-hop, inteligență.

RFID este ieftin și folosește foarte puțină energie. De aceea integrarea sa cu WSN este foarte utilă. Integrarea este posibilă în următoarele moduri [82, 83]:

- Integrarea etichetelor RFID cu senzorii: etichetele RFID cu capacități de detectare se numesc etichete senzor. Aceste etichete ale senzorilor detectează datele din mediu și apoi cititorul RFID poate citi aceste date detectate din etichetă. În astfel de cazuri, se utilizează protocoale RFID simple, unde există o singură comunicație hop. Tehnologiile de detectare RFID pot fi clasificate în continuare pe baza necesității de putere a etichetelor senzorilor.
- Integrarea etichetelor RFID cu nodurile WSN: capacitățile de comunicare ale etichetelor senzorilor sunt limitate la un singur salt. Pentru a-și extinde capacitățile, eticheta senzorului este

echipată cu un transceiver fără fir, puțină memorie Flash și capabilități de calcul, astfel încât să poată iniția comunicarea cu alte noduri și dispozitive wireless. Nodurile pot fi utilizate în acest mod pentru a forma o rețea fără fir mesh. În astfel de rețele, etichetele senzorilor pot comunica între ele pe o gamă largă (prin hamei intermediari). Cu capacități de procesare suplimentare la un nod, se reduce cantitatea netă de date comunicate și, respectiv, eficiența energetică a WSN.

- Integrarea cititoarelor RFID cu nodurile WSN: acest tip de integrare se face și pentru a crește gama de cititoare de etichete RFID. Cititoarele sunt echipate cu emițătoare-receptoare și microcontrolere fără fir, astfel încât să poată comunica între ele și, prin urmare, datele etichetei pot ajunge la un cititor, care nu se află în raza acelei etichete. Acesta profită de comunicarea multihop a dispozitivelor de rețea fără fir cu senzori. Datele de la toate cititoarele RFID din rețea ajung în cele din urmă la un gateway central sau o stație de bază care procesează datele sau le trimite către un server la distanță.

Aceste tipuri de soluții integrate au multe aplicații într-un set divers de domenii, cum ar fi securitatea, asistența medicală, producția ș.a.

Soluții cu rețele ZigBee

WSN-urile moderne constau dintr-un număr mare de dispozitive de detectare, care sunt ieftine și conectate între ele folosind comunicații de putere redusă, cum ar fi IEEE 802.15.4 sau transmițătoare ZigBee [84]. WSN se diferențiază funcțional de colecția obișnuită de dispozitive de detectare prin capacitățile sale de rețea, care permit cooperarea, coordonarea și colaborarea între activele de detectare [84]. În plus, în loc să trimită datele brute către nodurile responsabile de fuziune, nodurile de detectare își folosesc capacitățile de procesare pentru a efectua calcule locale simple și pentru a transfera doar datele necesare și procesate parțial [85].

Tehnologia ZigBee are multe avantaje fața de tehnologiile standard, cum este menționat în [86] despre capacitatea mare de a economisi energie în baterie și este capabilă să susțină un număr mare de noduri într-o rețea. Capacitatea de a extinde rețeaua este simplă și prezintă o securitate ridicată pentru utilizator; simplă de implementare – firmware-ul este actualizat de fiecare dată când este programat; număr redus de fire de conexiune; este o tehnologie pentru utilizare la nivel global. În lucrarea [81] se prezintă rezultatul simulării pentru trimiterea de date prin rețeaua de comunicații folosind tehnologia ZigBee, Fig. 1.7.

ZigBee este o specificație definită în IEEE 802.15.4 pentru o gamă de protocoale de comunicații destinate unui număr mare de dispozitive (până la ~ 65000) dintr-o rețea, cu un consum redus de energie, astfel încât dispozitivele să poată funcționa mult timp folosind doar o

sursă mică de energie, cum ar fi o baterie. Topologia de bază a rețelei ZigBee este mesh, dar poate funcționa pe alte tipuri de rețele, cum ar fi stea sau copac.

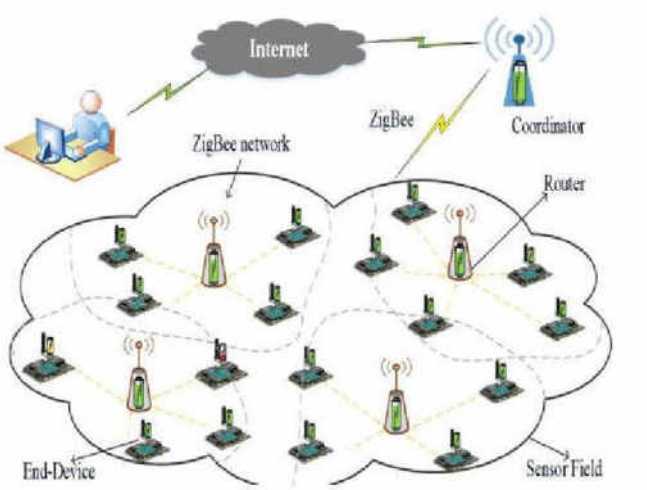


Fig. 1.7. Un model de sistem WSN care utilizează comunicații ZigBee [81]

Avantajul esențial al ZigBee este consumul redus de energie, utilizarea eficientă a lățimii de bandă și este o soluție ideală pentru IoT. Dezavantaj este costul mare al dispozitivelor, care se datorează licenței pentru a le produce. ZigBee folosește de obicei un radio de 2,4 GHz, integrând totul cu un microcontroler și memorie pentru stocarea instrucțiunilor și datelor. Costul ridicat se datorează cerințelor procesului de validare a sistemelor ZigBee și calificărilor acestora. Rata de transmisie standard este redusă – aproximativ 250 kbps, frecvența 2,4 GHz și o distanță de 10-20 m. Protocoalele actuale ZigBee acceptă două moduri de operare în rețea, cu bază radio activă și pasivă. În cel activ, routerele transmit periodic semnale radio pentru a-și confirma prezența în rețea, astfel încât nodurile să poată fi în modul de repaus în intervalele dintre semnalele de baliză, reducând astfel puterea consumată. În Fig. 1.8 este prezentată diagrama a unei rețele ZigBee.

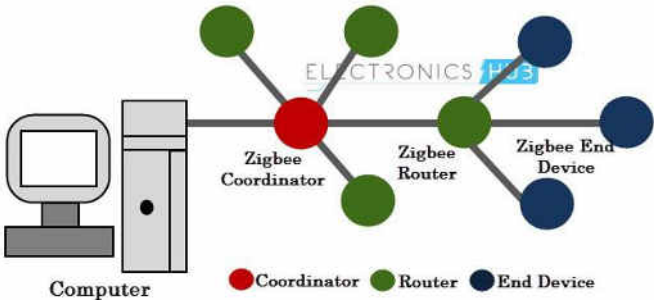


Fig. 1.8. Rețeaua ZigBee și componentele sale [87]

Tehnologia ZigBee a fost proiectată în conformitate cu standardele IEEE, pentru dezvoltarea de dispozitive care pot fi atât industriale, cât și de uz casnic. Tehnologia IEEE 802.15.4 ZigBee [88] este pe larg utilizată în diverse domenii de aplicație, după cum urmează:

- Automatizare și control case inteligente – se referă la gestionarea automatizării caselor, în care tehnologia ZigBee poate fi utilizată pentru controlul de la distanță a tuturor dispozitivelor electronice, cum ar fi lumini, alarme de securitate, comutatoare, uși, aer condiționat și alte dispozitive destinate pentru uz casnic [73];
- Contorizare inteligentă – se aplică în rețeaua de senzori fără fir pentru comunicarea între furnizorii de acasă și marii furnizori de servicii precum distribuitorii de energie electrică, distribuitorii de apă potabilă etc., facilitând astfel colectarea facturilor și îmbunătățind considerabil eficiența serviciilor lor [89];
- Îngrijirea sănătății – definește o interacțiune între medic și pacient. Utilizarea tehnologiei ZigBee pentru ca medicul să aibă mai multe informații sau date relevante despre pacientul său în timp real, evitând astfel faptul că pacientul să se deplaseze la spital de multe ori și să stea în rândurile lungi la medic. În așa mod, medicul poate avea date precum tensiunea arterială, pulsul, nivelul glucozei, temperatura corpului ș.a. [90];
- Controlul procesului industrial – prin utilizarea tehnologiei ZigBee a fost îmbunătățită semnificativ și industria de fabricație. Astăzi este posibil să existe mașini care funcționează 100% cu senzori fără fir, reducând astfel utilizarea firelor electrice pentru comunicații, dar și pentru electrificarea mașinilor în sine [91];
- Aplicații de telecomunicații – dispozitivele sunt conectate prin ZigBee și sunt tehnologiile utilizate în comunicații, cum este cazul telefoanelor mobile și computerelor. Acest lucru se datorează faptului că, pentru ca dispozitivele ZigBee să fie conectate, este necesar să se trimită informații către coordonator, care, la rândul său, trimite informații către un computer și de la computer la IoT [92];
- Agricultura Inteligentă – multe lucruri care pot fi explorate în ZigBee pentru agricultură, deoarece această zonă folosește mașini mari, motiv pentru care atunci când producția este pe scară largă, are nevoie de agricultură mecanizată, iar senzorii sunt esențiali pentru monitorizare, controler și manipularea acestui echipament. Senzorii pot face calcule reale despre calitatea solului, umiditatea, presiunea, cantitatea de producție de apă necesară pentru irigare, manipularea irigării și utilizarea dronelor pentru pulverizarea controlată de senzori la distanță. Atât de mult, încât ne putem gândi și la colectare și stocare, senzorii ar oferi informații despre succesul colecției și în timp real în care produsele pot rămâne în depozitare [93, 94];
- Aplicații militare – fenomenul atacurilor teroriste sau alte acțiuni malefice desfășurate de persoane ciudate au fost înregistrate în mai multe țări din întreaga lume, unde, în majoritatea cazurilor, originea și locația lor nu sunt cunoscute. Cu toate acestea, utilizarea tehnologiei

ZigBee a venit pentru a spori serviciile acestei zone de acțiune, deoarece este posibil să se recunoască trupele inamice în cazul atacurilor de natura menționată anterior [95];

- Mentenanță rețea electrică – în prezent, întreținerea lămpilor de iluminat și revizuirea contoarelor rezidențiale sunt efectuate de un tehnician, care se deplasează la locație pentru a lua datele și să urce pe piloni pentru a evalua problema. Această sarcină contravine politicilor de siguranță a muncii, care sunt din ce în ce mai prezente în companiile mari și cu legi din ce în ce mai exigente. Este de remarcat faptul că va exista o pierdere inutilă de timp pentru negocierea accesului la proprietate. Proiectul urmărește apoi instalarea dispozitivelor ZigBee pe echipamentul menționat. Dispozitivele ZigBee Router ar fi instalate pe fiecare pol al rețelei, care ar comunica cu centrala electrică prin hamei, iar dispozitivele finale ZigBee ar fi instalate pe contoare de acasă [96].

Comunicații CAN pent4u Automotive

În ultimele trei decenii, rețeaua CAN [97] a devenit un protocol de comunicații de facto utilizat în aplicațiile auto și de automatizare [98, 99, 100, 101]. Cu toate acestea, în acest timp, au avut loc creșteri semnificative în dimensiunea, scara și complexitatea atât a sistemelor electronice, cât și a sistemelor de automatizare [101]. Protocoalele CAN în prezent au ajuns la limitele lor [98, 100]. Unele rețele CAN folosesc acum comunicarea bazată pe TDMA pentru a ajuta la îndeplinirea constrângerilor în timp real. În timp ce această formă de control al accesului media aduce mai multe avantaje de actualitate, studiile au ilustrat efecte negative asupra fiabilității transmisiei, iar dublarea mesajelor poate ajuta la creșterea fiabilității [102]. În ciuda apariției unor scheme de comunicații precum FlexRay [103] și a unei utilizări sporite a tehnologiilor Ethernet comutate, CAN este cel mai probabil să coexiste cu alte protocoale, datorită popularității sale, simplității relative și ușurinței de implementare [98, 99, 100, 101]. CAN a fost inițial destinat să permită comunicarea declanșată de evenimente între nodurile nesincronizate din aplicațiile auto [97, 98, 99, 104]. Din diverse motive, unele rețele CAN folosesc scheme de comunicare cu divizare în timp (TDMA) implementate în principal ca extensii de protocol bazate pe hardware sau software [98, 105, 106]. Aceasta este văzută ca o încercare de a crește predictibilitatea comunicării, de a îndeplini constrângeri stricte în timp real, de a simplifica mecanismele de redundanță a canalelor și de a maximiza debitul la utilizări ridicate ale rețelei. Adoptarea schemelor TDMA nu este singura metodă propusă pentru a obține aceste beneficii. CAN se folosește pentru schemele de îmbunătățire a fiabilității și a transferului de date declanșat de evenimente native [107, 98].

1.2 Abordări actuale în modelarea sistemelor electronice distribuite

1.2.1 Concepte arhitecturale în sistemele electronice distribuite tip IoT

Nu există un consens unic asupra arhitecturii pentru IoT. Diferite cercetări au propus diferite arhitecturi în acest scop. Cea mai populară este *arhitectura cu trei straturi* [108, 109, 110], Fig. 1.9. Aceasta a fost introdusă în primele etape ale cercetării în acest domeniu, straturile fiind de percepție, rețea și aplicație:

- Stratul de percepție este stratul fizic, care are senzori pentru detectarea și colectarea informațiilor despre mediu prin parametri fizici sau identifică alte obiecte inteligente din mediu.
- Stratul de rețea este responsabil pentru conectarea la alte lucruri inteligente, dispozitive de rețea și servere. Caracteristicile sale sunt utilizate pentru transmiterea și procesarea datelor senzorilor.
- Stratul aplicației este responsabil de furnizarea serviciilor specifice aplicației către utilizator. Este definit de diverse aplicații, în care IoT poate fi implementat, de exemplu, case inteligente, orașe inteligente, sănătate inteligentă ș.a.

Arhitectura în trei straturi definește ideea principală a IoT, dar nu este suficientă pentru cercetarea asupra IoT, este necesara analiza aspectelor mai fine ale IoT. Există mult mai multe *arhitecturi în straturi* propuse în literatură. Una dintre acestea este *arhitectura cu cinci straturi*, care include adițional straturile de procesare și de business [108, 109, 110, 111]. Cele cinci straturi sunt straturi de percepție, transport, procesare, aplicație și business (vezi Fig. 1.9).

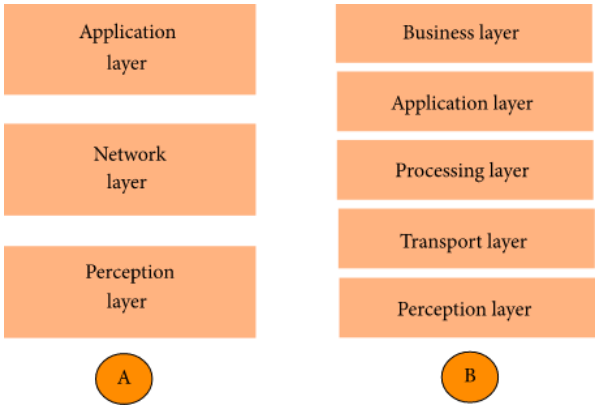


Fig. 1.9 Arhitectura IoT (A: trei straturi) (B: cinci straturi) [27]

Rolul straturilor de percepție și aplicație este același cu *arhitectura în trei straturi*, rolul celor trei straturi rămase fiind:

- Stratul de transport transferă datele senzorului din stratul de percepție în stratul de procesare și invers, prin rețele precum wireless, 3G, LAN, Bluetooth, RFID și NFC;
- Stratul de procesare este, cunoscut sub numele de stratul middleware. Stochează, analizează și procesează cantități uriașe de date care provin din stratul de transport. Poate gestiona și

furniza un set divers de servicii pentru straturile inferioare. Folosește multe tehnologii, cum ar fi baze de date, cloud computing și module de prelucrare a datelor mari;

- Stratul de business gestionează întregul sistem IoT, inclusiv aplicațiile, modelele de afaceri și de profit și confidențialitatea utilizatorilor.

O altă arhitectură propusă de Ning și Wang [112] este inspirată de straturile de procesare din creierul uman. Este inspirat de inteligența și capacitatea ființelor umane de a gândi, simți, aminti, a lua decizii și a reacționa la mediul fizic. Este alcătuit din trei părți. Prima parte este creierul uman, care este analog cu unitatea de procesare și gestionare a datelor sau cu centrul de date. Al doilea este măduva spinării, care este analogă rețelei distribuite de noduri de procesare a datelor și gateway-uri inteligente. În al treilea rând, este rețeaua de nervi, care corespunde componentelor de rețea și senzorilor.

1.2.2 Servicii de comunicare în sisteme distribuite

Resursele software, care oferă servicii și capabilități comune aplicațiilor în afara celor oferite de sistemul de operare, se numesc middleware. Middleware îi ajută pe dezvoltatori să construiască eficient aplicații, oferind soluții pentru gestionarea datelor, servicii de aplicații, mesageria, API de autentificare și gestionare. Acționează ca țesutul conjunctiv dintre aplicații, date și utilizatori [113]. Pentru organizațiile cu medii multi-cloud și containerizate, middleware-ul poate eficientiza dezvoltarea și rularea aplicațiilor la scară largă.

Calculul omniprezent este nucleul IoT, ceea ce înseamnă încorporarea computerului și a conectivității în toate lucrurile din jurul nostru. Interoperabilitatea acestor dispozitive eterogene necesită standarde bine definite, însă standardizarea este dificilă din cauza cerințelor variate ale diferitelor aplicații și dispozitive. Pentru astfel de aplicații eterogene, soluția este de a avea o platformă middleware, care va abstractiza detaliile lucrurilor pentru aplicații, adică va ascunde detaliile lucrurilor inteligente. Middleware acționează ca o punte software între lucruri și aplicații. Trebuie să ofere serviciile necesare dezvoltatorilor de aplicații [114], astfel încât aceștia să se poată concentra mai mult pe cerințele aplicațiilor decât pe interacțiunea cu hardware-ul de bază. Pentru a rezuma, middleware-ul abstractizează hardware-ul și oferă o interfață de programare a aplicațiilor (API) pentru comunicare, gestionarea datelor, calcul, securitate și confidențialitate [27]. Provocările, care sunt abordate de orice middleware IoT [114, 115, 116], sunt următoarele:

- Interoperabilitate și abstracții de programare – pentru a facilita colaborarea și schimbul de informații între dispozitive eterogene, diferite tipuri de lucruri pot interacționa ușor între ele cu ajutorul serviciilor middleware. Interoperabilitatea este de trei tipuri: rețea, semantică și sintactică. Interoperabilitatea rețelei se ocupă de protocoale de interfață eterogene pentru

comunicația între dispozitive și izolează aplicațiile de complexitatea diferitelor protocoale. Interoperabilitatea sintactică asigură că aplicațiile ignoră faptul ca datele au diferite formate, structuri și codificări. Interoperabilitatea semantică se ocupă de abstractizarea sensului datelor într-un anumit domeniu. Este inspirat în mod vag de rețeaua semantică.

- Identificarea și gestionarea dispozitivelor – această caracteristică permite dispozitivelor să fie la curent cu toate celelalte dispozitive din cartier și cu serviciile oferite de acestea. În IoT, infrastructura este în mare parte dinamică. Dispozitivele trebuie să își anunțe prezența și serviciile pe care le furnizează. Soluția trebuie să fie scalabilă, deoarece dispozitivele dintr-o rețea IoT pot crește. Majoritatea soluțiilor din acest domeniu sunt inspirate în mod vag de tehnologiile web semantice. Middleware-ul oferă API-uri pentru a lista dispozitivele IoT, serviciile și capacitățile acestora, precum și API-uri pentru a descoperi dispozitive pe baza capacităților lor. Orice middleware IoT trebuie să efectueze echilibrarea încărcării, să gestioneze dispozitivele în funcție de nivelul de energie a bateriei și să raporteze utilizatorilor problemele legate de dispozitive.
- Scalabilitate – se așteaptă ca un număr mare de dispozitive să comunice într-o configurație IoT. Mai mult, aplicațiile IoT trebuie să se extindă din cauza cerințelor din ce în ce mai mari. Acest lucru ar trebui gestionat de middleware prin efectuarea modificărilor necesare atunci când infrastructura se extinde.
- Big Data și analize – senzorii IoT colectează de obicei o cantitate imensă de date. Este necesar să fie analizate toate aceste date în detaliu. Ca rezultat, o mulțime de algoritmi de date mari sunt folosiți pentru a analiza datele IoT. Mai mult, este posibil ca, din cauza naturii slabe a rețelei, unele dintre datele colectate să fie incomplete. Este necesar să se țină seama de acest lucru și să se extrapoleze datele utilizând algoritmi sofisticați de învățare automată.
- Securitate și confidențialitate – aplicațiile IoT sunt în mare parte legate de viața personală a unei persoane sau de o industrie. Problemele de securitate și confidențialitate trebuie abordate în toate aceste medii. Middleware-ul necesită mecanisme integrate pentru a aborda astfel de probleme, împreună cu autentificarea utilizatorului și implementarea controlului accesului.
- Servicii cloud – cloud este o parte importantă a unei implementări IoT. Majoritatea datelor despre senzori sunt analizate și stocate într-un nor centralizat. Este necesar ca middleware-ul IoT să ruleze fără probleme pe diferite tipuri de nori și să permită utilizatorilor să profite de cloud pentru a obține informații mai bune din datele colectate de senzori.
- Detecția contextului datelor colectate de la senzori este necesară pentru a extrage contextul prin aplicarea diferitelor tipuri de algoritmi. Contextul poate fi ulterior utilizat pentru furnizarea de servicii sofisticate utilizatorilor.

Există multe soluții middleware disponibile pentru IoT, care abordează una sau mai multe dintre problemele menționate anterior. Toate acestea susțin interoperabilitatea și abstractizarea, care sunt cele mai importante cerințe ale middleware-ului. Câteva exemple sunt: Oracle's Fusion Middleware, OpenIoT [117], MiddleWhere [118] și Hydra [119]. Middleware-urile pot fi clasificate pe baza proiectării lor [116], după cum urmează:

- Bazat pe evenimente – toate componentele interacționează între ele prin evenimente. Fiecare eveniment are un tip și câțiva parametri. Evenimentele sunt generate de producători și primite de consumatori. Aceasta poate fi privită ca o arhitectură de publicare / abonare, în care entitățile se pot abona pentru anumite evenimente și pot fi notificate pentru evenimentele de interes;
- Orientat pe servicii – middleware-urile orientate pe servicii se bazează pe arhitecturi orientate pe servicii (SOA – Service Oriented Architecture), în care există module independente care oferă servicii prin interfețe accesibile. Un middleware orientat spre servicii consideră resursele ca furnizori de servicii. Acesta combină sub-servicii pentru un set de servicii utilizate de aplicații. Conține un depozit de servicii, unde serviciile sunt publicate de furnizori. Consumatorii pot descoperi servicii din depozit și apoi se pot lega de furnizor pentru a accesa serviciul. Middleware-ul orientat spre servicii trebuie să aibă suport pentru runtime, pentru serviciile de publicitate de către furnizori și suport pentru descoperirea și utilizarea serviciilor de către consumatori. Hydra [119] este un middleware orientat spre servicii. Încorporează numeroase componente software, care sunt utilizate în gestionarea diverselor sarcini necesare dezvoltării aplicațiilor inteligente. Hydra oferă interoperabilitate semantică, utilizând tehnologii web semantice. Sprijină reconfigurarea dinamică și autogestionarea;
- Orientat către baza de date – în această abordare rețeaua de dispozitive IoT este considerată ca un sistem de baze de date relaționale virtuale. Baza de date poate fi apoi interogată de aplicații, folosind un limbaj de interogare. Există interfețe ușor de utilizat pentru extragerea datelor din baza de date. Această abordare are probleme cu scalarea datorită modelului său centralizat;
- Semantic – middleware-ul semantic se concentrează pe interoperarea diferitelor tipuri de dispozitive, care comunică utilizând diferite formate de date. Încorporează dispozitive cu diferite formate de date și ontologii și le leagă pe toate într-un cadru comun. Cadrul este utilizat pentru schimbul de date între diverse tipuri de dispozitive. Pentru un format semantic comun sunt necesare adaptoare pentru comunicarea între dispozitive, deoarece pentru fiecare dispozitiv sunt necesare adaptoare pentru maparea standardelor la un standard abstract [120]. Într-un astfel de middleware semantic [121] este introdus un strat semantic, în care există o mapare de la fiecare resursă la un strat software pentru resursa respectivă. Straturile software comunică

apoi între ele, folosind un limbaj reciproc inteligibil. Această tehnică permite mai multe resurse fizice să comunice, chiar dacă acestea nu implementează sau înțeleg aceleași protocoale;

- Specific aplicației – acest tip de middleware este utilizat în mod special pentru un domeniu de aplicație pentru care este dezvoltat, deoarece întreaga arhitectură a acestui software middleware este reglată pe baza cerințelor aplicației. Aplicația și middleware-ul sunt strâns legate. Acestea nu sunt soluții cu scop general.

Noua tehnologie IoT conectează un număr masiv de obiecte inteligente, produse, dispozitive inteligente și oameni. În această tendință, toate lucrurile menționate ar putea fi organizate în grupuri sau sisteme și să coopereze în rezolvarea unor sarcini specifice, care oferă reciproc informații și servicii, implicând un proces tipic de comunicare. Comunicarea este actul de a transfera informații între lucruri ca entități ale unui sistem mare și joacă un rol esențial în funcționarea sistemului. Există o mulțime de tehnologii și tehnici contemporane, care implementează procesul de comunicare pentru transferul de informații între entități. Un fapt interesant este că aproape toate tehnologiile de comunicare sunt inspirate de comunicarea interpersonală, adică comunicarea dintre oameni.

Message Queuing Telemetry Transpor (MQTT) este un protocol special conceput pentru comunicarea „de la mașină la mașină”. Un exemplu bun de referință este instrumentul de rețea socială online Twitter, unde utilizatorii trimit și primesc postări scurte numite twit-uri. Oamenii care doresc să împărtășească informațiile pe care le publică și cei interesați de partajarea cuiva se abonează la tweet-urile sale [122]. Există multe caracteristici pentru această comunicare în rețeaua socială, dar principalul este sistemul de mesaje publicare-abonare.

Există multe asemănări între Twitter și unul dintre cele mai populare protocoale utilizate pentru aplicațiile IoT – MQTT, Fig. 1.10. Protocolul MQTT rulează pe TCP / IP și are o dimensiune a pachetului de date cu un cost redus minim (> 2 octeți), astfel încât consumul sursei de alimentare este suficient de mic [123]. MQTT este soluția perfectă pentru aplicațiile IoT, fiind proiectat pentru dispozitive cu lățime de bandă redusă. MQTT este un protocol simplu de mesagerie, care permite trimiterea, citirea și publicarea de date de la nodurile senzorilor ș.a. Comparativ cu Twitter, este același lucru, dar pentru dispozitive. Protocolul MQTT utilizează o arhitectură de publicare / abonare. Protocolul MQTT este bazat pe evenimente și permite transmiterea mesajelor către client [124]. Dispozitivele sunt ca utilizatorii Twitter care trimit mesaje către toți urmăritorii, un dispozitiv publică un mesaj către toți abonații săi. La fel ca un utilizator Twitter, care primește tweet-uri de la toți oamenii pe care îi urmărește, un dispozitiv primește mesaje de la toate dispozitivele la care este abonat [125].

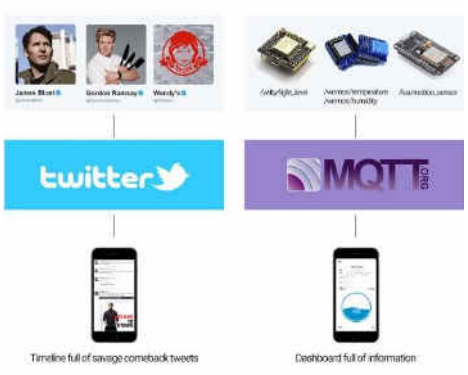


Fig. 1.10. MQTT – Twitter pentru dispozitive [125]

FiWare este un cadru de middleware IoT foarte popular, promovat de UE. A fost conceput pentru a ține cont de orașele inteligente, logistica și analiza magazinelor. FiWare conține un corp mare de cod, module reutilizabile și API-uri, la care au contribuit mii de dezvoltatori FiWare. Orice dezvoltator de aplicații poate prelua un subset al acestor componente și își poate crea aplicația IoT [27]. O aplicație tipică IoT are mulți producători de date (senzori), un set de servere pentru procesarea datelor și un set de actuatoare. FiWare se referă la informațiile colectate de senzori ca informații de context. Acesta definește API-urile REST generice pentru a captura contextul din diferite scenarii. Toate informațiile de context sunt trimise unui serviciu numit broker de context. FiWare oferă API-uri pentru a stoca contextul și pentru a-l interoga. Orice aplicație se poate înregistra ca un consumator contextual și poate solicita informații brokerului contextual. FiWare acceptă paradigma de publicare-abonare. Ulterior, contextul poate fi furnizat sistemelor folosind adaptoare de context, al căror rol principal este de a transforma datele (contextul) pe baza cerințelor nodurilor de destinație. FiWare definește un set de API-uri SNMP, prin intermediul cărora se poate controla și configura comportamentul dispozitivelor IoT. Aplicațiile țintă sunt API-uri furnizate pentru a analiza, interoga și extrage informațiile colectate de la brokerul de context. Cu API-uri de vizualizare avansate, este posibil de a crea și implementa într-un timp redus aplicații cu o diversitate mare de funcționalități.

OpenIoT este o inițiativă populară Open Source și are la bază șapte straturi diferite [117]. La nivelul cel mai de jos este stratul fizic, care colectează date de pe dispozitivele IoT și face o anumită preprocesare a datelor. Are diferite API-uri pentru interfață cu diferite tipuri de noduri fizice și pentru a obține informații de la acestea [27]. Următorul strat este stratul virtualizat, care are trei componente. Prima componentă este programatorul, care gestionează fluxurile de date generate de dispozitive, le atribuie resurse și se ocupă de cerințele QoS. Componenta de stocare a datelor gestionează stocarea și arhivarea fluxurilor de date. Componenta de furnizare a serviciilor procesează fluxurile și are mai multe roluri. Aceasta combină fluxurile de date, le preprocesează

și urmărește unele statistici asociate acestor fluxuri, cum ar fi numărul de solicitări unice sau dimensiunea fiecărei cereri. Stratul superior, stratul aplicației, are trei componente: definirea cererii, prezentarea cererii și configurarea. Componenta de definire a cererii facilitează crearea de cereri pentru a fi trimise către senzorii IoT și straturile de stocare. Poate fi folosită pentru preluarea și interogarea datelor. Componenta de prezentare a cererii creează un amestec de date prin emiterea de interogări diferite către stratul de stocare și, în cele din urmă, componenta de configurare permite configurarea dispozitivelor IoT.

Robot Operating System (ROS) este o altă arhitectură de publicare / abonare implementată într-un sistem de operare a unui robot, o colecție de programe care permit unui utilizator să controleze cu ușurință operațiunile mobile ale unui robot. În esență, ROS este un middleware anonim de publicare / abonare, care transmite mesaje cu comunicații asincrone. Unele module vor publica un set de subiecte, în timp ce altele se abonează la acest subiect. Când sunt publicate date noi, abonații pot afla despre actualizări și pot acționa asupra lor. Comunicarea este implementată utilizând o abordare de transmitere a mesajelor, care îi obligă pe dezvoltatori să se concentreze pe logica de stabilire de a interfețelor [126]. ROSCORE este un serviciu care oferă informații de conexiune către noduri, astfel încât acestea să poată transmite mesaje unul către celălalt. Fiecare nod se conectează la ROSCORE la pornire pentru a înregistra detalii despre fluxurile de mesaje pe care le publică și fluxurile la care dorește să se aboneze. Când apare un nou nod, ROSCORE îi oferă informațiile de care are nevoie pentru a forma o conexiune directă peer-to-peer cu alte noduri care publică și se abonează la aceleași subiecte de mesaje. Fiecare sistem ROS are nevoie de un ROSCORE care rulează, deoarece, fără acesta, nodurile nu pot găsi alte noduri [126].

Automobilismul este unul dintre cele mai critice domenii de siguranță. Un sistem auto tipic constă din mai multe ECU-uri care comunică pe busul de CAN. Stivele de software oferă suport pentru calcule și comunicații, inclusiv sarcini de aplicație, middleware, drivere și periferice de magistrală. În scopuri de comunicare, sarcinile citesc semnalele de intrare la activare și își scriu ieșirile în variabile partajate la sfârșit. Unele sarcini ale aplicației citesc datele de intrare de la un senzor, calculează rezultatele intermediare care sunt trimise prin rețea către alte sarcini și, în final, o altă sarcină, care se execută pe un nod la distanță, generează ieșirile ca rezultat al calculului [126]. Operațiile de citire / scriere cu variabile partajate ar putea fi comparate cu arhitectura de publicare / abonare pentru comunicarea prin rețeaua vehiculului.

1.2.3 Dezvoltarea bazata pe modele

În prezent, sistemele încorporate își cresc continuu complexitatea hardware și software, deplasându-se către soluții cu un singur cip, System-on-Chip (SoC). În același timp, nevoile de

piață ale proiectelor SoC cresc rapid, impunând presiuni pe timpul de ieșire pe piață. Ca urmare a acestor tendințe inovatoare, industriile de semiconductori adoptă fluxuri de co-proiectare hardware / software, unde sistemul proiectat este reprezentat la un nivel înalt de abstracție ca un set de macro-blocuri reutilizabile hardware și software [127].

Sistemele încorporate sunt utilizate în mai multe domenii, cum ar fi robotică, mașini inteligente, rețele de senzori fără fir, compuse din dispozitive minuscule încorporate. Acestea pot fi definite ca sisteme de senzori distribuiți cu o infrastructură configurată automat. Pentru a garanta fiabilitatea și performanța în dezvoltarea acestor sisteme, utilizarea metodelor formale reprezintă o problemă ambițioasă. Metodele formale permit specificarea la nivel înalt, evită ambiguitatea și oferă tehnici de verificare. Cu toate acestea, trecerea de la descrierea la nivel înalt la implementarea concretă rămâne un pas informal și predispus la erori. Formalizarea acestei etape și automatizarea ei este un domeniu atractiv de cercetare [128].

Ingineria Bazată pe Modele (Model Driven Engineering – MDE) se concentrează pe un proces de dezvoltare care face ca modelele să fie obiectul central de dezvoltare. Software-ul final livrat este derivat de la aceste modele și nu este niciodată modificat direct. Instrumentele sunt capabile de a transforma modelele proiectate în implementări cu limbajele dorite. Generarea automată de cod și Ingineria Bazată pe Modele sunt foarte promițătoare, incluzând reducerea de defecte și productivitate sporită [129, 130].

O tendință impunătoare a domeniului IoT este Internetul a Orice – Internet of Everything, ceea ce înseamnă că orice obiect sau lucru care ne înconjoară este un potențial participant la rețeaua mondială. Ca regulă, un lucru sau obiect conectat la Internet este un sistem încorporat, ce presupune adeseori, încorporarea unei unități de procesare de performanță mică într-un obiect de uz zilnic. Întrucât numărul de lucruri conectate la Internet crește exponențial, reutilizarea soluțiilor existente este un punct semnificativ pentru a face față cererii mari pentru noul tip de lucruri care trebuie conectate la rețea. Pe lângă reutilizarea în sine, abordarea arhitecturală corectă care are reutilizabilitate reprezintă conceptul cheie. Reutilizabilitatea presupune generalizarea unor funcționalități comune pentru a fi utilizate masiv în dezvoltarea dispozitivelor încorporate, și respectiv crearea soluțiilor în baza lor.

Un impact considerabil în optimizarea procesului de dezvoltare o au instrumentele de proiectare automatizată. Ingineria automatizată de dezvoltare implică eforturi computaționale în procesul de inginerie software [131]. Scopul efortului este automatizarea parțială sau completă a acestor activități, crescând semnificativ atât calitatea, cât și productivitatea. Această automatizare include studiul tehnicilor de construcție, înțelegere, adaptare și modelare a ambelor domenii, atât

a produselor software, cât și a proceselor de dezvoltare. Abordări ce țin de automatizarea procesului de proiectare au fost aplicate în multe domenii ale ingineriei software [132].

Instrumentele pentru crearea Limbajelor de Modelare Specifice Domeniului (Domain Specific Modeling Languages – DSML), devin tot mai acceptate în industria de dezvoltare a software-ului pentru a dezvolta soluții specifice pentru probleme specifice. Metamodelul de bază al instrumentelor este de o importanță crucială, deoarece servește ca bază pentru toate etapele ulterioare, cum ar fi generarea de coduri sau gestionarea programată a datelor modelului. Prin urmare, accesul la metamodel trebuie să fie foarte simplu, iar consumul de memorie mic [133].

Vizualizarea grafică a modelelor vine în ajutorul perceperii vizuale a sistemelor și facilitează perceperea în ansamblu a elementelor arhitecturale ale resurselor programabile dezvoltate. O modalitate simplă de generare și vizualizare a diagramelor de model este Graphviz, un software liber. Vizualizarea modelelor cu Graphviz este o modalitate de reprezentare a informațiilor structurale ca diagrame ale graficelor și rețelelor abstracte. Are aplicații importante în tehnologiile de rețea, bioinformatică, inginerie software, proiectare de baze de date și web, învățare automată și în interfețe vizuale pentru alte domenii tehnice [134]. Notarea Graphviz este destul de simplă în esență, fapt pentru care este selectată în multe aplicații pentru prezentarea grafică a rezultatelor sub forma de diagrame.

Un exemplu excelent al unei arhitecturi bine definite este conceptul AUTomotive Open System ARchitecture (AUTOSAR). Aceasta stă la baza dezvoltării noilor module în sistemele auto și reprezintă cea mai nouă tendință auto din întreaga lume. Standardul AUTOSAR definește arhitectura și metoda de referință pentru dezvoltarea sistemelor software auto și furnizează limbajul de descriere (metamodelul) pentru modelele lor arhitecturale. Totodată, AUTOSAR specifică modulele arhitecturale și funcționalitatea stratului middleware cunoscut sub numele de Software de bază (BSW) pentru platformă [135].

Utilizarea sistemelor de inginerie electrică (EE) în vehicule crește rapid. Aceste sisteme sunt foarte complexe. O mașină modernă poate avea aproximativ 100 de ECU-uri care comunică între ele prin intermediul rețelei CAN [136] și aproximativ un milion de linii de cod. Software-ul pentru ECU-uri este din ce în ce mai sofisticat și dependent de hardware, ceea ce face procesul de dezvoltare consumator de timp sau excesiv de scump. Pentru a rezolva această problemă, AUTOSAR face hardware-ul și software-ul aplicației ECU independente și permite reutilizarea componentelor software similare. În consecință, acest lucru reduce efortul, timpul și costul procesului de dezvoltare software [137]. AUTOSAR este un standard industrial emergent pentru sistemele software auto [138]. Dezvoltarea sa este determinată de necesitatea de a aborda complexitatea crescândă a sistemelor auto moderne și de a reduce costurile de dezvoltare atunci

când se introduc noi caracteristici bazate pe software. AUTOSAR este organizat într-o *arhitectură modulară în straturi* și se bazează pe un proces de dezvoltare centrat pe componente / compoziții care standardizează schemele de modelare și denumire din sistem, inclusiv componente, interfețe, tipuri de date și funcționalități cu fire de execuție [139].

Funcționalitățile generice, cum ar fi memoria, comunicarea sau securitatea, sunt realizate în Software-ul de Bază (Basic Software – BSW), în interiorul stivelor în straturi a componentelor, distribuite între trei straturi generice, cum ar fi Stratul de Servicii (Service Layer – SRVL), Stratul de Abstracție ECU (ECU Abstraction Layer – ECAL) și Stratul de Abstracție MCU (MCU Abstraction Layer – MCAL). Funcționalitățile care nu pot fi generalizate sunt definite ca driver de dispozitiv complex (Complex Device Driver – CDD) Fig. 1.11.

Conceptul abstractizează echipamentul electric printr-un strat specific numit Mediu de rulare (Runtime Environment – RTE), astfel încât Software-ul de Aplicație (Application Software – ASW) să nu ia în considerare tipul de echipament cu care interacționează sau localizarea resurselor necesare.

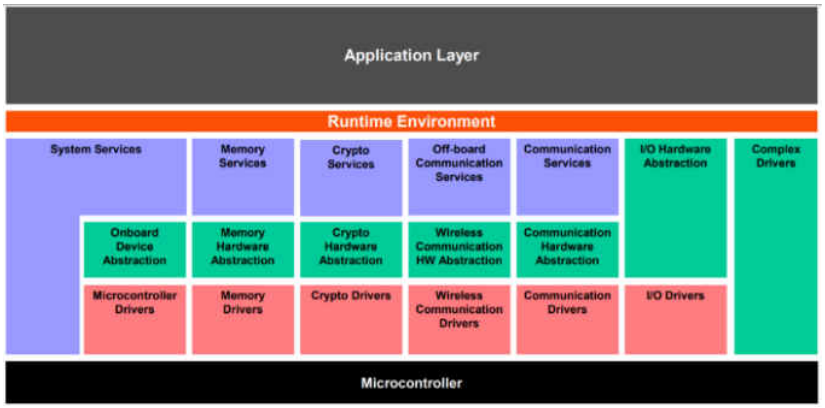


Fig. 1.11. Arhitectura în straturi AUTOSAR [135]

Responsabilitatea de a asigura funcționalitățile și conectivitatea necesare este transferată către componentele din BSW al platformei Fig. 1.12.

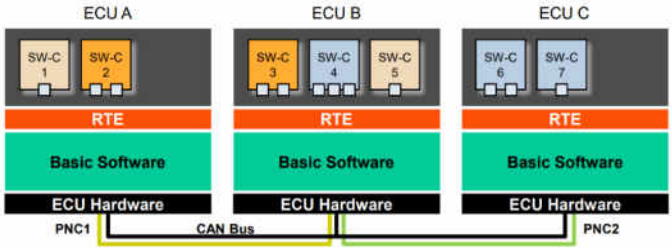


Fig. 1.12. Conceptul de interacțiune componentă AUTOSAR [135]

Conceptul menționat permite aplicației să acceseze resursele situate pe alte echipamente ca și cum ar fi parte a echipamentului pe care rulează, ceea ce se datorează unei abstracții și mai mari, în care toate straturile RTE formează un concept de Magistrală Virtuală de Funcționare (Virtual

Functional Bus – VFB). Acest concept permite realizarea de aplicații distribuite, în care componentele pot fi distribuite pe diferite ECU, Fig. 1.13.

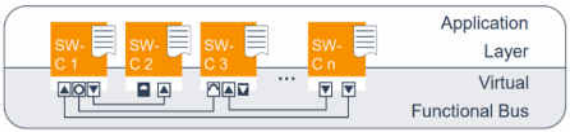


Fig. 1.13. Conceptul AUTOSAR Virtual Functional Bus [135]

O rețea auto în vehicule constă din mai multe domenii de operațiuni în cadrul acesteia [140]. Fiecare dintre domenii constă din propria unitate de control intern pentru procesarea sa. Fiecare ECU din rețeaua de echipamente a vehiculului este echipat, ca regulă, cu cel puțin un MCU [141]. Acest microcontroler este echivalent cu un mini computer și este echipat cu propria memorie internă, capacitate de procesare și este denumit System on Chip (SoC) [140]. În cadrul unui vehicul, numărul de ECU-uri conectează domeniile funcționale ale vehiculului la finalizarea sarcinilor centralizate și cooperate și sunt distribuite în mediul fizic al autovehiculului [141]. Tipul de memorie a acestui MC permite executarea instrucțiunilor în cadrul cipului în sine. Această simplitate atomică face ca rețeaua din interiorul vehiculului să fie echipată cu mai multe ECU-uri pentru a controla domeniile funcționale din vehicul [142].

O rețea de ECU-uri în vehicul este grupată pentru a aduce toate operațiunile și comenzile unui automobil cu controlul intenționat al șoferilor pe vehicul. În loc de un singur procesor, arhitectura distribuită a rețelei de ECU-uri ajută șoferul și pasagerii unui automobil în multe moduri funcționale [143]. Un ECU este un element de procesare, care primește intrări de la senzori ai domeniului funcțional corespunzător, procesează cu funcționalitățile sale interne, intervenind cu reacții de remediere prin intermediul actuatorilor [140, 144].

Componentele senzor/actuator este un tip specific de componentă software AUTOSAR pentru achiziții de la senzor și controlul actuatorului. Localizarea componentelor SW senzor/actuator este deasupra RTE din motive de integrare, Fig. 1.14 [135].

Conform AUTOSAR, modelul de proiectare, design pattern, al senzorului / actuatorului descrie modul de gestionare a senzorilor sau actuatorilor conectați la un ECU în contextul unei arhitecturi generale. Modelul de proiectare a senzorului / actuatorului se concentrează pe următoarele aspecte conceptuale:

- independența software-ului de aplicație față de senzorii și actuatorii conectați la un ECU;
- cod reutilizabil între diferiți senzori și actuatori;
- diferite modele de cooperare în partajarea codurilor (partajarea software-ului) ș
- implementarea funcționalității pe diferite ECU-uri [140].

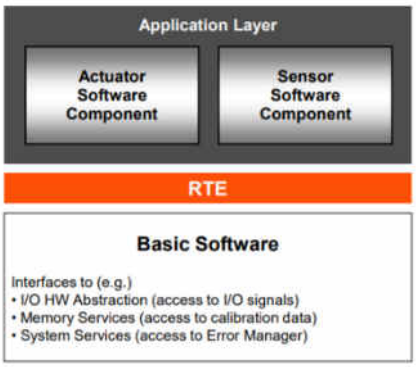


Fig. 1.14. Componenta senzor/actuator în arhitectura AUTOSAR [135]

Stiva de componente senzor / actuator este definită ca trei straturi, care sunt legate de driverul de dispozitiv virtual, driverul de dispozitiv hardware și interfața electrică, care corespund componentei SW, abstractizarea ECU și abstractizarea MCU. Vezi Fig. 1.15 pentru un exemplu.

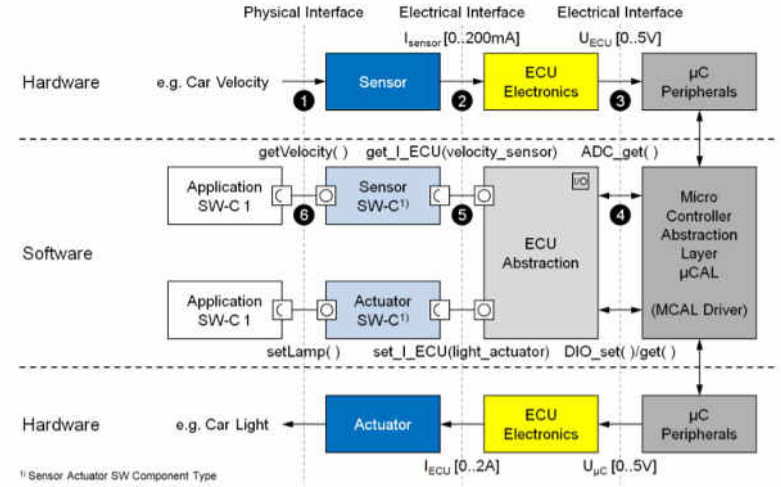


Fig. 1.15. Exemplu de stivă a senzorului actuatorului [145]

Fiecare ECU din rețeaua din vehicul are un control specific, colectează intrarea de la diferiți senzori ai vehiculului, procesează cu setul său predefinit de instrucțiuni și trimite ieșirea către diferiții actuatori ai vehiculului. Acest lucru are ca rezultat computerizarea operațiunilor de domeniu funcțional ale vehiculului. Domeniul funcțional al unui automobil constă din diferite funcții de control electronice, cum ar fi controlul tracțiunii vehiculului, controlul stabilizării (mișcării / pornirii), monitorizarea caroseriei în vehicul pentru siguranță activă (în timpul deplasării și opririi), sincronizare multimedia / infotainment, navigare GPS, în exterior comunicare mondială, control activ/pasiv al siguranței și sisteme de identificare/raportare a defecțiunilor [145].

Caracteristicile produsului auto sunt realizate în conformitate cu cerințele utilizatorului, legislației și performanței, cu un anumit nivel de personalizare și cerințe nefuncționale. Diagnosticul a agravat și mai mult complexitatea procesului de dezvoltare a ECU [146, 147]. Acest scenariu implică necesitatea de a adopta noi metodologii și instrumente pentru a ușura dezvoltarea,

reducând costurile, resursele și timpul de comercializare. Întrucât industria automobilelor este o piață extrem de dinamică și agresivă, un timp scurt pe piață poate fi decisiv dacă un nou model va avea succes sau nu. În mediul actual de dezvoltare este dificilă reutilizarea software-ului încorporat pentru automobile care este deja dezvoltat pentru un ECU [148]. Prin introducerea AUTOSAR, un tip de arhitectură software standardizată, reutilizarea poate fi îmbunătățită, precum și timpul și costurile pot fi economisite, iar AUTSOAR poate fi o soluție de gestionat. AUTOSAR [149] urmărește un standard industrial pentru unitățile de control distribuite în vehicule [150].

Într-un flux de lucru AUTOSAR, s-ar putea găsi o varietate de instrumente oferite de diferiți producători pentru a efectua o parte specifică a procesului, Fig. 1.16. Toate aceste soluții, pot fi împărțite în trei clase de instrumente [150]:

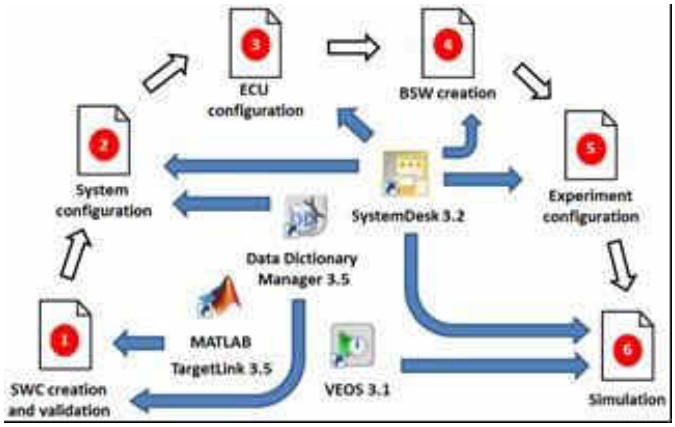


Fig. 1.16. Flux de lucru pentru validarea virtuală AUTOSAR [150]

- IDE, BMT și C – corespund dezvoltării componentelor software (Software Component – SWC), utilizând Instrumente de Modelare a Comportamentului (Behavioral Modelling Tool – BMT), de exemplu: Simulink, TargetLink sau scrise manual în limbaj C. Fiecare SWC este compus dintr-un fișier de descriere XML AUTOSAR (.ARXML) și fișierul dvs. respectiv *.c. Aceasta corespunde stratului superior din nivelul AUTOSAR [150].
- Instrumente de proiectare la nivel de sistem – responsabile de dezvoltarea sistemelor și subsistemelor. Unele acțiuni ar putea fi executate, cum ar fi conexiunile SWC și maparea ECU, topologia hardware, gestionarea comunicațiilor magistralei, maparea elementelor de date etc. Rezultatele acestui instrument sunt: Fișierele de descriere a sistemului tip *.ARXML și Fișierele de configurare a ECU de tip *.ARXML pentru fiecare ECU în sistem [150, 151].
- Instrumente de configurare software de bază – responsabile de configurarea fiecărui ECU individual și generarea fișierelor *.hex înregistrate în microcontrolerele ECU [150, 153].

Ciclul de proiectare al AUTOSAR definește fișiere de configurare necesare pentru maparea SWC-urilor la ECU-uri interconectate în rețeaua unui vehicul prin intermediul unei tehnologii, cum este prezentat în Fig. 1.17.

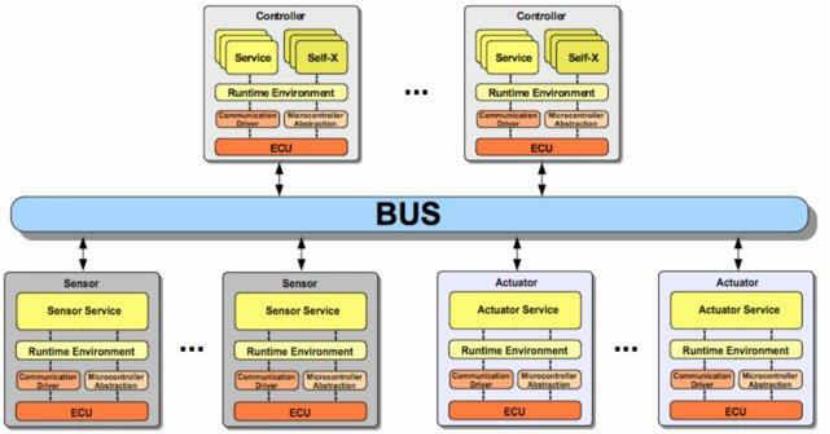


Fig. 1.17. Rețea de noduri, senzori și actuatoare [151]

Middleware-ul organic definește, două fișiere de configurare: unul pentru serviciile aplicațiilor și unul pentru fiecare nod. Fișierul de configurare al unui nod descrie resursele disponibile și senzorii sau actuatorii atașați. Fișierul de configurare al aplicației descrie cerințele de resurse ale serviciilor și dependențelor, care descriu cerințele unui serviciu sub formă de alte servicii, senzori sau actuatori. Descrierile de configurație sunt date în XML și pot fi ușor extinse pentru a îndeplini noile cerințe. O descriere detaliată a limbajului de descriere a configurației și a schemei XML poate fi găsită în [145]. În cele ce urmează este prezentat un exemplu pentru definiția unui actuator [151], Fig. 1.18. Serviciile și posibilele dependențe pot fi definite, după cum urmează în Fig. 1.19.

```
<actuator id="windowlift_back_left"
  name="windowlift"
  class="de.uau.sik.actuator.impl.WindowLift">
  <resource name="motor">
    <value name="speed" unit="rotation/s">5</value>
    <value name="lifting_force" unit="gramm">
      500</value>
  </resource>
</actuator>
```

Fig. 1.18. Exemplu pentru definiția unui actuator în XML [151]

Serviciile și dependențele odată definite în instrumentul de gestionare al platformei AUTOSAR, sunt evaluate în timpul configurării ulterioare, introducând recomandări, constrângeri și alte facilitați de platformă în procesul de proiectare și gestionare a proiectului.

```
<service id="window_lifters" amount="1"
  name="WindowLiftSystem" priority="2"
  class="de.uau.sik.service.impl.WindowLift">

  <resource name="cpu">
    <value name="frequency"
       unit="quantiSpeed">10</value>
    </resource>
    <resource name="ram">
      <value name="size" unit="kB">512</value>
    </resource>
    <resource name="programSize">
      <value name="size" unit="kB">100</value>
    </resource>

    <dependency type="actuator" amount="1"
       priority="1">windowlift_back_left
    </dependency>
    ...
    <dependency type="sensor" amount="4"
       priority="1">push_button
    </dependency>
    <dependency type="service"
       priority="1">aircondition
    </dependency>
</service>
```

Fig. 1.19. Servicii și dependențe definite cu XML [151]

Integrare și validare

Siguranța funcțională a sistemelor încorporate în special pentru domeniul auto este o problemă cheie în timpul procesului de dezvoltare. Lucrarea [152] propune o abordare pentru integrarea și testarea sistemelor critice de siguranță prin utilizarea tehnicilor de modelare a sistemelor. Combinația a două limbaje de modelare de ultimă generație într-un proces dedicat de dezvoltare în mai multe limbi oferă o legătură directă între toate etapele procesului de dezvoltare, permițând astfel verificarea și validarea eficientă a siguranței deja în faza de modelare.

Validarea prin teste de integrare în sistemele încorporate este un proces de verificare a funcționalității, performanței și fiabilității sistemului software care rulează pe un dispozitiv hardware încorporat. Testarea de integrare implică combinarea mai multor componente care au fost testate individual și evaluarea modului în care acestea interacționează între ele. Testarea de integrare este o etapă importantă în dezvoltarea sistemelor electronice distribuite, deoarece acestea implică adesea o complexitate ridicată și o interdependență între componentele software și hardware. Testarea de integrare permite dezvoltatorilor să verifice dacă sistemul funcționează corect ca un întreg, fără a afecta funcționalitatea sau performanța componentelor individuale.

1.3 Concluzii la capitolul 1

În baza analizei literaturii în domeniul modelării sistemelor electronice distribuite, deducem următoarele concluzii:

1. Actualitatea domeniului modelării sistemelor electronice distribuite sunt reprezentate prin multitudinea de sisteme de tip IoT pe plan internațional și confirmate prin creșterea exponențială a numărului de obiecte interconectate în rețele de acest tip.
2. Modelarea sistemelor electronice distribuite reprezintă un factor decisiv în controlul complexității interacțiunilor și problemelor pe care le soluționează.
3. Multitudinea de soluții existente în domeniul modelării sistemelor distribuite, chiar dacă vizează diferite tehnologii, folosesc principii similare de interacțiune, cum ar fi client-server sau publicare-abonare.
4. În multe dintre soluțiile care și-au demonstrat fiabilitatea pe parcursul anilor este utilizată abordarea *arhitecturală în straturi*, cum ar fi nivelele pentru Interconexiuni Sisteme Deschise (Open Systems Interconnection – OSI) în domeniul rețelelor de comunicare sau AUTOSAR în domeniul Automotive.
5. Sursa primară de informație este întotdeauna mediul fizic, iar pentru a fi transferată în interiorul unui sistem digital, informația trece printr-un *lanț complex de funcții de transfer*.
6. Fiecare interacțiune între componentele din sistem poate fi privită ca un proces de comunicare intern, iar sistemele în întregime pot fi privite ca o totalitate de componente interconectate cu o abstractizare de tip lanț de comunicare.
7. Asigurarea funcționarii stabile din punct de vedere electric ale sistemelor este una crucială, iar mecanismele de diagnoză și protecție au un rol important în această problemă.
8. Luând în considerație dinamica creșterii numărului de aplicații ale sistemelor distribuite, reutilizarea modelelor care au fost validate în timp este esențială, efortul principal fiind axat pe dezvoltarea de noi aplicații prin extinderea soluțiilor existente deja.
9. Un factor important asupra timpului de elaborare al aplicațiilor îl reprezintă automatizarea proiectării prin utilizarea modelelor configurabile, efortul de proiectare revine definiției de configurații a modelelor reutilizabile, iar procesul devine unul preponderent descriptiv.
10. Instrumentele de modelare, configurare și generare automată reduce esențial efortul de dezvoltare al aplicațiilor și efortul de control al complexității sistemului.
11. Starea actuală a cercetărilor în domeniul modelării sistemelor electronice distribuite are o dinamică exponențială și prin urmare este în continuă căutare de soluții inovatoare pentru a

face față cererii impuse de tehnologizarea tuturor domeniilor umanității, cum ar fi agricultura, electrocasnice, construcții, transport, industrie ș.a.

În baza acestor concluzii, s-au formulat scopul și obiectivele lucrării:

Scopul lucrării: Proiectarea sistemelor electronice distribuite cu generarea automată a configurației în baza dezvoltării de modele și produse program, utilizând concepte arhitecturale în straturi și *metode de organizare a fluxurilor de informație în lanțuri de comunicare*, scop atins prin ***următoarele obiective***:

- Elaborarea conceptului de arhitectură generică a unui sistem electronic încorporat prin evidențierea domeniilor de interacțiune și definirea de componente în straturi.
- Elaborarea unui concept comun pentru achiziția informației și acționare asupra mediului, asociind diagnozele cu fluxul de achiziții, iar reacțiile de protecție cu acționarea.
- Elaborarea unui concept pentru organizarea fluxurilor de informație al componentelor de interacțiune cu mediul și interacțiunea între dispozitivele electronice;
- Definirea modelului de componentă configurabilă a sistemului, cât și a metodei de adaptare a soluțiilor externe preluate și reutilizate în cadrul sistemului.
- Elaborarea metodologiei de modelare a sistemelor electronice distribuite prin modelare arhitecturală cu metamodele și generarea automată a configurației și a codului sursă.
- Dezvoltarea produsului program pentru modelare și definirea automată a configurațiilor bazată pe metamodele și generare de cod sursă de platformă.
- Crearea resurselor online de componente reutilizabile ca bază de soluții pentru dezvoltare de sisteme în scopul optimizării timpului alocat modelarii de sisteme în bază de resurse existente.
- Dezvoltarea aplicaților prin metoda de modelare arhitecturală a sistemelor electronice distribuite și procesare a fluxurilor de informații pentru validarea metodei propuse.
- Analiza rezultatelor obținute cu scopul de adaptare la noi domenii de aplicații.

2. METODE DE MODELARE A SISTEMELOR ELECTRONICE DISTRIBUITE

2.1 Considerente de modelare arhitecturală pentru sisteme electronice distribuite

La nivel de sistem, facem abstracție de la orice detalii de implementare, fie mecanice, electrice, software sau de orice alt tip. Considerăm sistemul doar ca un set de componente și distribuirea funcției complexe de transfer al sistemului între componentele ce îl compun.

La cel mai înalt nivel de abstractizare, sistemul ar putea fi privit ca un dispozitiv care preia informații din mediu înconjurător, le prelucrează și reacționează pe baza factorilor interni și de mediu. În cadrul lucrării [154] sunt prezentate două aspecte ale sistemului: structural – Fig. 2.1 și funcțional – Fig. 2.2

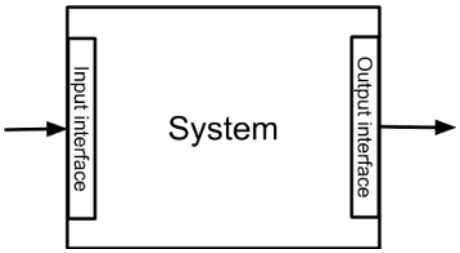


Fig. 2.1 Sistem ca componentă ce interacționează cu mediul [154]

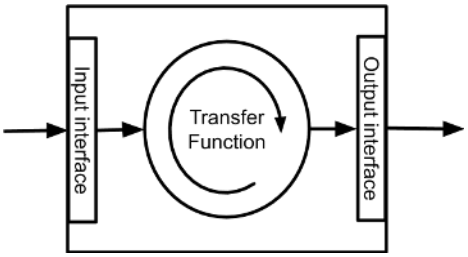


Fig. 2.2. Componentă reprezentată de funcția ei de transfer [154]

Din punct de vedere structural, sistemul reprezintă o combinație de componente interconectate, Fig. 2.3. Pentru un dispozitiv IoT, astfel de componente ar putea enumera un senzor, un actuator, interacțiunea cu utilizatorul, comunicarea, baza de date și managementul energiei sau componentele aplicației specifice dispozitivului, așa cum este prezentat în Fig. 2.7.

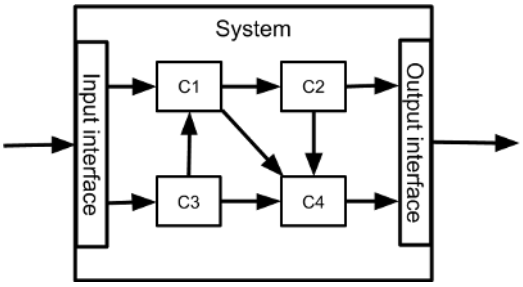


Fig. 2.3. Sistem ca o combinație de componente care îl compun [154]

Din punct de vedere funcțional, sistemul reprezintă combinația de funcții de transfer, prin care un semnal din interfața mediului se propagă prin diferite funcții de transfer până la atingerea ieșirii sistemului, oferind semnale înapoi către mediu, așa cum se arată în Fig. 2.4.

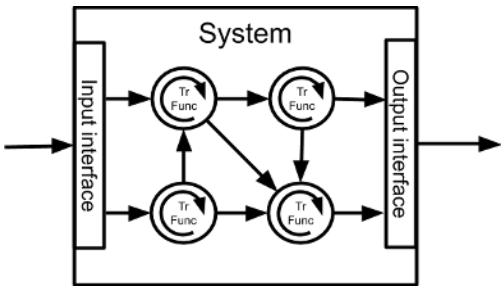


Fig. 2.4. Sistem ca o combinație de funcții de transfer care îl compun [154]

În contextul tezei *funcție de transfer* se consideră o abstractizare a unei funcționalități care poate fi reprezentată fie printr-o relație matematică fie printr-un comportament sau algoritm care produce un rezultat în dependență de parametri de intrare. În calitate de parametri de intrare pentru funcție pot servi parametri interni sau externi al sistemului, stări interne sau externe, semnale interne sau externe, incluziv timpul. Anumiți parametri pot servi ca și configurații ale funcționalității, ce permit ajustarea acesteia. Semnal se va considera orice evoluție a informației în cadrul purtătorului de informație indiferent de tehnologia prin care este realizat purtătorul.

Această abstractizare se aplică în toate domeniile implicate în realizarea sistemului, cum ar fi ingineria mecanică (ME), ingineria electronică și electrica (EE), ingineria software (SWE) sau de orice alt tip, aspectul principal constituind sistemul ca un set de componente interconectate și transformarea cu funcția de transfer a informației.

2.1.1 Modelul arhitecturii generice al sistemului electronic distribuit

În acest context introducem următoarea definiție – un *sistem electronic distribuit* este un sistem electronic integru, considerat ca un dispozitiv electronic întreg, părți componente ale căruia sunt detașate și distribuite în spațiul fizic, iar liniile de semnal de transfer al informației, sunt restabilite cu tehnologii de comunicare.

O explicație prin exemplu ar fi următoarea – presupunem ca inițial avem un dispozitiv electronic foarte complex realizat pe un singur cablaj imprimat. În cazul în care este nevoie de interacțiune cu mediul exterior la o distanța anumită, o parte a acestui cablaj este decupata/tăiată, asigurându-i-se condițiile normale de funcționare din punct de vedere electric. Liniile de semnal care în urma decupării au fost întrerupte, sunt transmise prin interfețe de comunicare cu fir sau fără fir prin diverse tehnologii de comunicare, în dependență de cerințele față de semnal sau sistem.

O rețea de dispozitive electronice este considerată drept un sistem electronic distribuit, în care rețeaua înlocuiește sau abstractizează conexiunile de semnal, inițial distribuite prin fir, pentru interconectarea componentelor, Fig. 2.5.

Sistemele IoT pot fi abstractizate ca un sistem de dispozitive electronice distribuite, care colectează informații de la intrare și generează reacții la ieșire [154]. O rețea IoT poate fi vizualizată ca parte a unui sistem electronic distribuit complex, în care rețeaua înlocuiește conexiunile prin cablu pentru transmiterea semnalului. O reprezentare grafică a acestui concept poate fi văzută în Fig. 2.5.

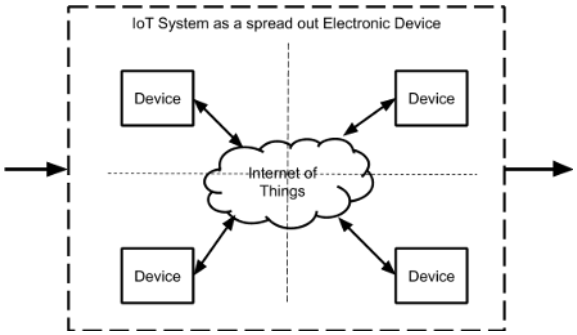


Fig. 2.5. Dispozitive electronice distribuite cu interconexiuni tip IoT [154]

Un dispozitiv electronic ca parte al unui sistem distribuit este reprezentat prin două aspecte: interacțiunile dispozitivului electronic distribuit și componentele ce compun dispozitivul: *Interacțiunile unui dispozitiv electronic distribuit* și *Componentele generice ale dispozitivului*.

Interacțiunile unui dispozitiv electronic distribuit – reprezintă primul aspect care este tratat din punctul de vedere al interacțiunilor cu componentele și factorii externi. Aceste interacțiuni reprezintă interacțiunea cu mediul înconjurător, interacțiunea cu utilizatorul și comunicarea între dispozitive, Fig. 2.6

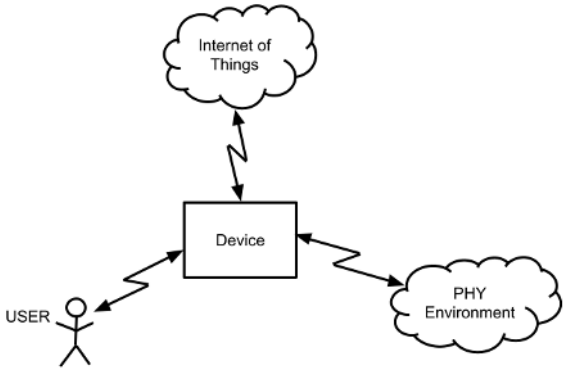


Fig. 2.6. Interacțiuni ale dispozitivului electronic distribuit

- mediul înconjurător (PHY) – este reprezentat de parametrii săi fizici, care urmează să fie achiziționați sau influențați de dispozitiv.

- utilizatorul (USER) – fie uman, fie inuman, care poate interveni cu introducerea sau urmărirea mesajelor sau acțiunilor de către dispozitiv.
- Internet of Things (IoT) – reprezintă toate tehnologiile de comunicare existente, care permit schimbul de date între dispozitive.

Aceste tipuri de interacțiuni impun constrângeri specifice acesteia, dar și metodologii și tehnologii specializate, multe dintre ele fiind tipice modului de interacțiunii, rezolvând probleme de comunicare, adaptare la mediul înconjurător și modalități de interacționare cu utilizatorul.

Componente generice ale dispozitivului – este cel de-al doilea aspect, care presupune gruparea funcționalităților de bază ale dispozitivului în componente generice, după rolul lor în sistem, definind o *arhitectură generică* a dispozitivului.

Funcționalitățile sistemului sunt grupate în componente generice specifice, ca părți care pot se regăsesc în cadrul unui dispozitiv standard. Aceste componente conțin funcționalități în funcție de problema pe care o rezolvă, cum ar fi achiziție, acționare, interacțiune cu utilizatorul, comunicare, stocare de date, gestionare a energiei, platformă sau componente ale sistemului de operare. Funcționalitățile de aplicație fiind în afara acestei definiții, însă ele se dezvoltă în baza serviciilor pe care le oferă acest concept.

Având în vedere toate ipotezele, autorul tezei propune [155] următoarea diagramă arhitecturală a unui dispozitiv conectat pentru un sistem electronic distribuit de tip IoT, Fig. 2.7.

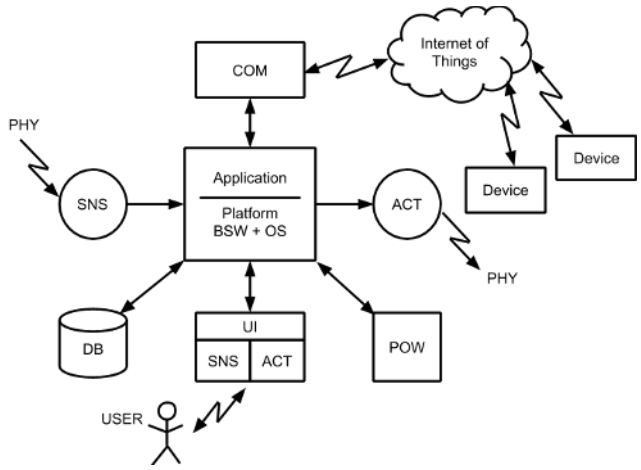


Fig. 2.7. Arhitectura generica a unui dispozitiv electronic IoT [155]

- SNS – senzor, reprezintă setul de componente implementate în Ingineria Mecanică (ME), Ingineria Electrică (EE) sau SW Engineering (SWE), implicate în achiziționarea și transformarea unui parametru fizic PHY al mediului în informații INFO interne ale sistemului.
- ACT – actuator, reprezintă setul de componente implementate în Ingineria Mecanică (ME), Ingineria Electrică (EE) sau SW Engineering (SWE), implicate în transformarea informațiilor INFO de control în acțiune în parametrul fizic (PHY) al mediului.

- UI – interacțiunea cu utilizatorul, reprezintă ansamblul tuturor componentelor de tip SNS sau ACT, specializate pentru interacțiunile umane sau inumane ale utilizatorilor.
- COM – comunicare, reprezintă ansamblul de componente realizate în Ingineria Mecanică (ME), Ingineria electrică (EE) sau SW Engineering (SWE), care intervin pentru a asigura transferul de informații între dispozitivele interconectate.
- DB – bază de date sau structuri de date, reprezintă setul de componente pentru stocare și acces la datele stocate pe termen scurt sau lung în cadrul dispozitivului sau acces la suportul de stocare la distanță. Acest domeniu ar putea fi identificat, la fel, ca memorie (MEM).
- POW – managementul energiei, reprezintă toate componentele care asigură funcționarea normală a dispozitivului din punct de vedere electric și gestionarea consumului de energie.
- BSW – este platforma care constă din sistemul de operare (OS) și serviciile disponibile pentru aplicația care rulează pe platformă, asigurând un nivel ridicat de abstractizare pentru dispozitiv.

Acest model de arhitectură generică al unui sistem electronic distribuit reprezintă o abstractizare simplificată a interacțiunilor sistemului, precum și abstractizarea componentelor funcționale generice. Componentele generice sunt caracteristice atât pentru un dispozitiv electronic, cât și pentru întregul sistem electronic distribuit ce realizează funcționalitatea complexă a sistemului.

2.1.2 Modelele arhitecturale în straturi ale componentelor generice al sistemului electronic distribuit

Urmând modelul de *arhitectură generic*ă pentru un sistem electronic distribuit, următorul pas este de a defini modelele arhitecturale pentru fiecare componentă din care este constituit sistemul. În acest scop se propune utilizarea conceptului propus de AUTOSAR și a conceptului de *arhitectură în straturi* al stivei de comunicare TCP/IP [154]. Se propune ca unele componente ale arhitecturii dispozitivului să fie acoperite de specificațiile de componente, conform arhitecturii AUTOSAR de către BSW [135], și anume OS, COM, MEM / DB, IO, CDD, vezi Fig. 1.11. Lista componentelor se va extinde printr-o extensie numită ESW – Extended Software, respectiv se propune un concept generic pentru așa-numita clasă de componente ESW pentru o *arhitectură în straturi* de tip AUTOSAR cu asociere HW / SW, definind soluții pentru stiva de componente senzor (SNS) și Actuator (ACT). În AUTOSAR, componentele de tip senzor și actuator sunt definite la nivel de aplicație, considerându-se specifice aplicației. Se prezintă o soluție generică reutilizabilă comună pentru toate tipurile de componente menționate în *arhitectura generică* a unui dispozitiv: senzor, actuator, comunicare și interacțiune cu utilizatorul Fig. 2.7. Reguli similare de stive de componente ar putea fi aplicate pentru toate cele trei tipuri de interacțiuni – mediul fizic,

utilizatorul și rețeaua Fig. 2.6. Același concept se poate aplica componentelor, precum managementul datelor sau managementul energiei.

Fiecare componentă constă din părți implementate de Ingineria Mecanică (ME), Ingineria Electrică (EE) sau Ingineria SW (SWE). Acestea sunt organizate într-o *arhitectură în straturi*, pentru a asigura interacțiunile între toate domeniile, pentru a facilita transferul sigur și calitativ al semnalului / informațiilor către aplicație și invers, de la aplicație la mediu, utilizator sau alte sisteme, Fig. 2.8.

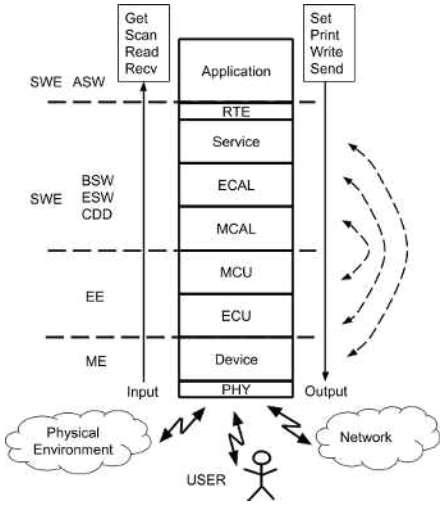


Fig. 2.8. Arhitectură în straturi pentru o componentă generică [154]

Conform conceptului prezentat, Fig. 2.8., în domeniul Software-ului, sunt definite multe straturi care abstractizează și furnizează servicii legate de straturile asociate din domeniul electric și mecanic. Aceste servicii sunt cunoscute sub numele de drivere de dispozitiv – biblioteci care acceptă echipamente periferice. În acest fel, conform diagramei din Fig. 2.8, se prezintă următoarele asociații:

- MCU vs. MCAL – MCU reprezintă toate perifericele hardware interne ale microcontrolerului, cum ar fi GPIO, TIMER, ADC, USART, SPI, EEPROM, WDT etc. MCAL (MCU Abstraction Layer) reprezintă serviciile software de suport ale perifericelor microcontrolerului. Printre resursele acestui strat pot fi furnizate funcționalități precum DigitalRead, DigitalWrite, PortRead, PortWrite, AnalogRead, AnalogWrite, SerialRead, SerialWrite, care sunt specifice pentru transferul de informații, dar și alte caracteristici specifice periferiilor.
- *ECU vs. ECAL* – nivelul ECU reprezintă componentele electronice situate pe placa microcontrolerului sau atașate la aceasta. ECAL fiind stratul de drivere pentru soluțiile electronice de pe ECU. Aceste componente electronice sunt părțile electrice ale componentelor din conceptul arhitectural al unui dispozitiv, Fig. 2.7, cum ar fi senzori, actuatori, cipuri de

memorie externe, dispozitive de interacțiune cu utilizatorii, module de comunicații, mecanisme de gestionare a energiei.

- *Device vs. Service* – unde Device reprezintă dispozitivul fizic în sine, căruia îi sunt asociate toate caracteristicile, conform specificației. Service reprezintă SW-ul de nivel superior al componentei. Este producătorul de servicii furnizate aplicației și asigură funcționalitățile de gestionare a componentei, precum și interacțiunea cu celelalte servicii din alte straturi.
- *PHY vs. RTE* – probabil nu este cea mai bună asociere, dar deoarece PHY reprezintă mediul fizic prin care dispozitivul interacționează cu mediul, RTE este cel prin care aplicația interacționează cu mediul, oferind funcții prefixate de Get, Scan, Read, Recv pentru a aduce informații din mediu și respectiv Set, Print, Write, Send pentru a transfera informații în mediu. Respectiv, putem presupune că RTE este abstracția față de PHY, mediul înconjurător.
- *Aplicație* – nu are un nivel asociat, aceasta reprezintă sistemul în sine. Comportamentul sistemului este implementat la nivel de aplicație. Aplicația poate fi implementată local pe un singur dispozitiv sau distribuită pe mai multe dispozitive. În mod similar, mai multe aplicații ar putea rula pe același dispozitiv, la fel ca și același dispozitiv poate face parte din mai multe sisteme. Similar conceptului de arhitectură în straturi AUTOSAR, Fig. 1.13.

O generalizare suplimentară se poate face considerând că interacțiunea cu utilizatorul și managementul energiei nu este altceva decât totalitatea de senzori și actuatori cu destinație specială, iar managementul datelor – o modalitate de transfer de date asociată comunicării, respectiv se obțin trei tipuri de componente în straturi – senzor, actuator și comunicare. Componentelor de comunicare în sisteme le revine direcționarea fluxurilor de informație prin sistem, interconectarea componentelor. Conceptual această idee este prezentată în Fig. 2.9.

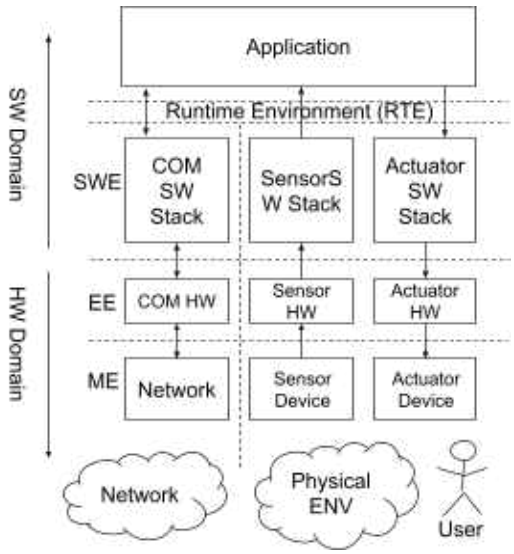


Fig. 2.9. Interacțiuni și domenii în conceptul de dispozitiv electronic distribuit [156]

2.2 Modelarea componentelor de tip senzor-actuator

Componentele din categoria senzor-actuator constituie elemente esențiale în sistemele electronice distribuite prin faptul că asigură interacțiunile cu mediul pentru a transfera informația din mediul real, fizic și a acționa asupra acestuia pentru a provoca o schimbare.

Metode de clasificare a componentelor de tip senzor-actuator

Pentru a asigura o flexibilitate înaltă a soluțiilor existente, dar și o reutilizare pe larg a acestora în definirea sistemelor, este necesar un studiu al componentelor introducând un set de clasificatori. Acești clasificatori vor permite evidențierea problemelor comune, similaritatea și discrepanțele între componente. Clasificarea va facilita luarea deciziilor privind tehnologiile aplicabile și soluțiile care pot fi reutilizate sau adaptate. Mai jos se propune o clasificare după următoarele criterii:

- Parametrii fizici achiziționați sau acționați – soluția hardware ar trebui să poată funcționa în mediul corespunzător, să fie sensibilă la parametrul fizic, în cazul senzorilor, și sa poată emite o acțiune în domeniul parametrului, în cazul actuatorilor. Metodele software ar trebui să ia în considerare proprietățile mediului fizic atunci când se definește modelul comportamental.
- Interfața fizică a semnalului electric – poate fi de diferit format, cum ar fi binar, analogic, digital sau bazat pe timp și implică constrângeri la selectarea sau adaptarea la microcontroler sau la alte componente electrice.
- Localizarea componentelor – ar putea fi locală sau globală, ceea ce înseamnă că componentele hardware sunt atașate la același ECU în cazul componentelor locale sau sunt accesibile de la distanță prin tehnologii de rețea specifice pentru cele globale.
- Ținta de funcționare – ar putea fi internă sau externă, ceea ce înseamnă că este detectare a unui efect intern în cazul senzorilor sau o acțiune în interiorul dispozitivului pentru actuatori. Pentru cazul de funcționare externă – funcționalitățile sunt relevante pentru parametrii fizici sau efectele din afara dispozitivului. Acest fapt implică considerații privind construcția mecanică, gestionarea energiei, sincronizarea, siguranța etc.
- Buclă prin mediul fizic – presupune existența unei bucle prin mediul fizic, astfel încât un senzor necesită un actuator pentru a funcționa sau invers – actuatorul are nevoie de detectare prin senzor. Un senzor poate fi activ sau pasiv. Pentru senzorul activ – este necesară o acțiune asupra mediului pentru a obține datele de interes. Un actuator se poate afla într-o buclă deschisă sau închisă. Pentru o buclă închisă, actuatorul necesită o colectare de răspuns de la mediu, pentru a oferi o acțiune precisă. Acest aspect are unul dintre cele mai semnificative impacturi asupra complexității modelului comportamental al componentelor.

2.2.1 Modelarea componentelor de achiziție și condiționare a semnalului de la senzori

Informațiile primare despre mediu au o stare fizică sau un efect fizic, în funcție de natura sa. De exemplu, luminozitatea este definită de un flux de fotoni; umiditatea – de concentrația moleculelor de apă în aer; temperatura – prin iradiere în infraroșu; sunetul – prin oscilații mecanice; distanța este exprimată în unități metrice etc. [157]. Reieșind din faptul că natura informațiilor primare este preponderent ne-electrică, adică este o natura ne-electrică a efectului măsurat, și faptul că echipamentele contemporane funcționează cu semnale electrice, este necesară o conversie a mărimilor neelectrice în cele electrice [154, 158, 159, 160]. Astfel, această conversie este funcția principală a senzorului.

La nivel de componentă – senzorul detectează modificările fizice care apar în mediul inconjurător și le transformă în mărimi măsurabile. Aceste mărimi sau valori neelectrice nu pot fi utilizate direct și este necesară o condiționare suplimentară cu funcții care sunt în responsabilitatea altei componente – traductorul. Traductorul convertește cantitatea neelectrică într-un semnal electric. Ca o concluzie, pentru a obține o dimensiune fizică exprimată în semnalul electric, sunt necesare două componente – un senzor conectat la un traductor. Deseori, ambele componente – senzorul și traductorul, sunt utilizate pentru a se referi la întreaga pereche, Fig. 2.10.

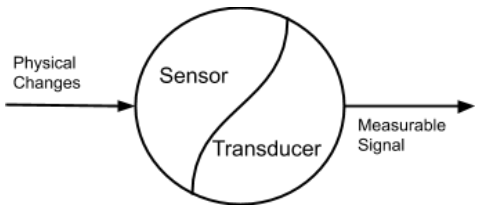


Fig. 2.10. Concept generic de senzor [154]

Valoarea colectată direct din elementul senzoristic este de obicei una brută și necesită condiționare înainte de a fi utilizată ca valoare fizică reală în cadrul sistemului, ca regulă, în resursele de program. Astfel, semnalul din momentul achiziționării de către senzor parcurge un drum lung printr-un *lanț de funcții de transfer*. Aceste funcții ar putea fi îndeplinite de ambele părți, în domeniul electronic, la fel ca în domeniul software. Există un senzor cu o interfață digitală care poate avea deja mecanismul de condiționare și poate fi implementat în traductor. Cu toate acestea, în cazul clasic sau în proiectarea traductorului în sine, condiționarea trebuie proiectată luându-se în considerare toate aspectele metodei de achiziție și mediul prin care se propagă semnalul purtător de informație.

Condiționarea semnalului se referă la manipularea unui semnal într-un mod care îl pregătește pentru următoarea etapă de procesare. Multe aplicații implică măsurători de mediu sau

structurale, cum ar fi temperatura și vibrațiile, de la senzori. În consecință, senzorii necesită condiționarea semnalului înainte ca un dispozitiv de achiziție de date să poată măsura semnalul în mod eficient și cu precizie înaltă [161]. Metodele de condiționare a semnalului sunt supuse atât senzorilor, cât și dispozitivelor de acționare și pot funcționa cu o mare varietate de semnale din sistemul proiectat.

O cale tipică pe care ar urma-o un semnal pentru o sursă de semnal analogică este de a parcurge domeniul electronic cu condiționare preliminară, după care urmează o condiționare în domeniul software, de exemplu, ca în Fig. 2.11.

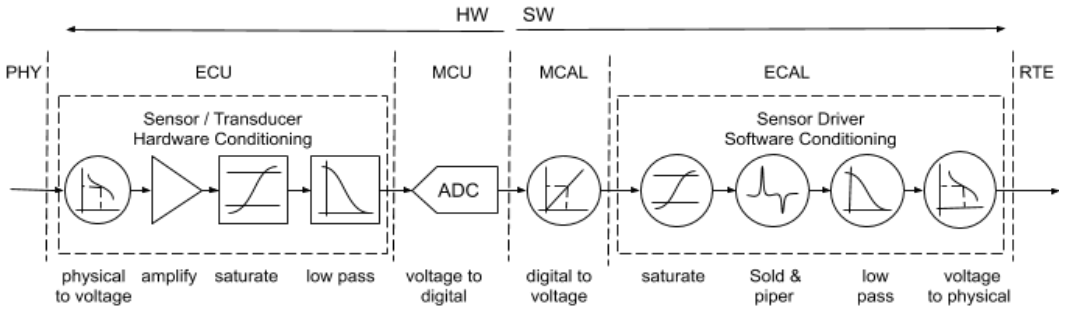


Fig. 2.11. Fluxul tipic de condiționare a semnalului de la senzor [154]

- Achiziționarea semnalului – reprezintă funcția de transfer care convertește parametrul fizic într-un semnal electric. Această funcție de transfer poate fi găsită în specificația tehnică a senzorului în cazul în care este unul industrial sau o curbă experimentală, care poate fi extrasă prin setarea condițiilor de mediu și măsurarea parametrului electric la ieșirea senzorului.
- Amplificarea / atenuarea și decalarea electrică – semnalul electric obținut poate avea nevoie de o amplificare pentru a putea vedea schimbarea valorii fizice aplicate senzorului la ieșirea electrică a senzorului, precum și adusă în intervalul valorilor acceptabile de către interfața microcontrolerului prin stabilirea unui decalaj, care este de obicei în intervalul 0-5 Volți. Amplificarea, precum și setarea de decalaj pe partea electrică, pot fi realizate cu metodologia funcțiilor cu amplificatoare operaționale. Atenuarea poate fi obținută cu circuite pasive, cum ar fi un divizor de tensiune sau, prin aceeași metodă, cu amplificatoare operaționale cu un coeficient de amplificare subunitar.
- Saturația electrică – este esențial să nu se admită semnale de intrare cu nivele de tensiune în afara intervalului acceptabil de microcontroler, care ar putea deteriora acest circuit. Presupunând că în faza de amplificare și decalaj sunt aduse la intervalul acceptabil pentru achiziționarea microcontrolerelor, valorile din afara acestui interval nu prezintă interes și pot fi saturate la valorile minime sau maxime ale intervalului. Realizarea acestui mecanism este

posibilă atât cu diode stabilizatoare de tip circuit simplu, cât și prin metodologia de efectuare a funcțiilor cu amplificatoare operaționale.

- Filtrarea electrică – se recomandă o filtrare preliminară în condiționarea electrică pentru a exclude influențele electromagnetice și perturbările de achiziție ale semnalului. De obicei, un Filtru Trece Jos (FTJ) și ar putea fi implicat în filtrarea frecvențelor înalte, specifice zgomotului alb, dar ar putea fi aplicat și un Filtru Trece Sus (FTS) sau Filtru Trece Bandă (FTB). Filtrul FTS ar putea exclude variația punctului de referință, decalarea semnalului. Un caz particular este aplicarea filtrului oprește bandă (Notch Filter), care exclude frecvențele induse de sursele de tensiune menajere 50-60 Hz. Filtrele electrice pot fi realizate fie cu elemente pasive sau elemente active, precum amplificatoare operaționale, fie cu circuite specializate.
- Conversie analog-digitală – presupune că semnalul analogic aplicat unui pin al microcontrolerului este convertit într-o valoare digitală, care poate fi recuperată cu tehnici de inginerie software, adică citind datele dintr-un registru în spațiul de adresă al perifericelor. Conversia analog-digitală (ADC) limitează rezoluția definiției valorii semnalului, precum și frecvența maximă de eșantionare la cea specifică convertorului. Caracteristicile unui ADC includ: precizia exprimată în numărul de cifre pe care le produce în funcție de valoare (de exemplu, convertorul ADC de 10 biți); viteza exprimată în conversii maxime pe secundă (de exemplu, 500 de conversii pe secundă); domeniul de măsurare exprimat în volți (de exemplu, 0-5V) [154]. Convertorul poate fi unul din interiorul microcontrolerului, permițând colectarea directă a semnalului analogic, unul extern cu interfața serială sau paralelă, comutat la interfețele respective ale microcontrolerului sau unul inclus în componenta senzorului, care furnizează semnal digital convertit. Multe module de ADC includ și un multiplexor, care permite conectarea mai multor senzori, ale căror date pot fi citite și convertite ulterior, dar această proprietate a convertorului implică și o scădere a frecvenței de eșantionare pe canal împărțit la numărul de senzori conectat [154].
- Conversia de la valoarea digitală RAW la valoarea de voltaj – implică operația inversă a conversiei analog-digitale, obținându-se astfel valoarea fizică a nivelului de tensiune la pinul microcontrolerului la care este aplicat semnalul. În multe cazuri, această conversie are o dependență liniară, în care ecuația liniei drepte poate folosi valorile minime și maxime din ADC, asociate cu tensiunile respective. De exemplu, 0..5V asociat cu intervalul de valori 0..1023 pentru convertorul cu o rezoluție de 10 biți. Din acest moment al evoluției semnalului, procesul de măsurare este terminat, iar valoarea tensiunii este transferată către o structură internă de date, astfel încât semnalul este gata să fie procesat cu metode software.

- Saturația semnalului software – este o funcție similară cu cea din domeniul electronic și prezintă limitarea valorii semnalului între o valoare minimă și maximă. Când în domeniul electronic saturația era un mecanism de protecție electrică a interfețelor electronice, în software, acest mecanism limitează câmpul de definiție a valorii pentru a minimiza efortul de calcul cu numere mari, care, totuși, nu are niciun interes deoarece sunt în afara domeniului de definiție.
- Filtrarea mediană – reprezintă filtrul care exclude impulsurile locale minime și maxime din semnal, cauzate, de exemplu, de factori cum ar fi vârfurile electromagnetice din cauza comutațiilor electrice din mediu. Filtrul median, numit și filtru statistic sau sare și piper, poate fi implementat prin sortarea pe o fereastră eșantion din semnalul de intrare și extragerea valorii din mijloc. În acest fel, minimele și maximele locale, după sortare, ajung la marginile vectorului sortat, iar în centru este cea mai probabilă valoare pentru eșantionul dat [162]. Recomandarea este ca fereastra eșantionului să aibă un număr impar de elemente pentru a facilita selectarea medianei, Fig. 2.12.

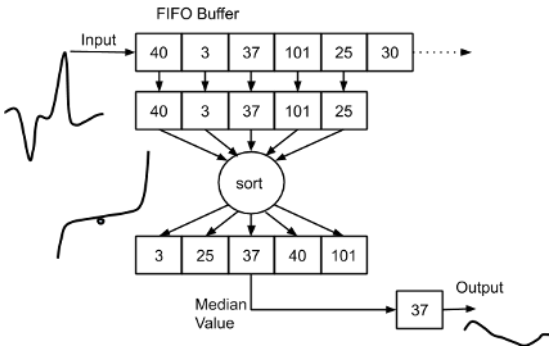


Fig. 2.12. Diagrama fluxului de date pentru implementarea filtrului median [154]

Filtru mediere ponderată – reprezintă un „Filtru Trece Jos" (FTJ) pentru excluderea zgomotului uniform, numit și zgomot alb. Media ponderată se aplică unei ferestre de eșantionare din semnalul stocat în buferul de intrare conform formulei (2.1).

$$Output = \frac{x_1 w_1 + x_1 w_1 + \ \ldots + x_k w_k}{w_1 + w_1 + \ \ldots + w_k} = \frac{\sum_{i=1}^{k} x_i w_i}{\sum_{i=1}^{k} w_i} \qquad (2.1)$$

Pentru eficiență în ceea ce privește performanța, se recomandă o fereastră de eșantionare minimă, care pentru microcontrolerele de performanță redusă poate fi redusă chiar și la ultimele două valori achiziționate. Dacă ponderile sunt egale se poate reduce la media aritmetică simplă.
unde x_i reprezintă valorile semnalului în fereastra de eșantionare, iar w_i sunt ponderile valorilor din rezultatul obținut, Fig. 2.13.

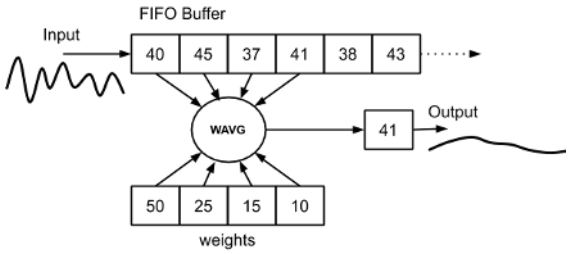


Fig. 2.13. Diagrama fluxului de date pentru filtrul mediere ponderat [154]

Conversia valorii voltaj la valoarea fizică – reprezintă procesul invers de conversie a tensiunii la valoarea fizică asociată senzorului, care convertește o valoare fizică în tensiune, dar în valoare ca informație internă utilizată în procesarea software-ului. Ca bază de calcul pentru această conversie este dependența de intrare / ieșire, care poate fi găsită în specificația tehnică a senzorului sau obținută experimental. În cazul dependenței liniare, ecuația canonică a liniei drepte (2.2) este utilizată cu două puncte de referință specifice senzorului.

$$\frac{x - x_1}{x_1 - x_2} = \frac{y - y_1}{y - y_2} \tag{2.2}$$

unde (x_1, y_1) și (x_2, y_2) sunt punctele de referință 1 și 2 pentru conversia cu aproximare lineară asociate ca y – tensiune, Volt, x – dimensiune fizică, PHY, conform Fig. 2.14.

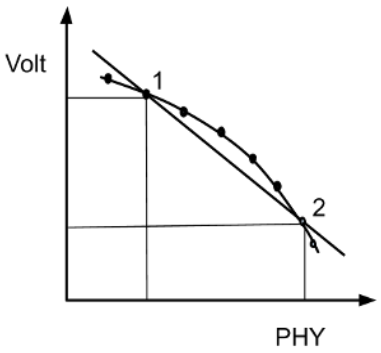


Fig. 2.14. Exemplu de funcție de transfer neliniară pentru un senzor [154]

În cazul dependenței neliniare, ca în Fig. 2.14, curba experimentală este împărțită în segmente mici conform tabelelor colectate experimental, cu o aproximări liniare, pe care s-ar putea aplica și ecuația conică a liniei drepte (2.2) pentru conversie.

2.2.2 Modelarea componentelor de acționare și condiționare semnal pentru acțiune asupra mediului

Urmând definiția din Technopedia [163], un actuator este un dispozitiv care deplasează sau controlează un mecanism. Un actuator transformă un semnal de comandă în acțiune mecanică, cum ar fi un motor electric. Actuatorul poate fi bazat pe mijloace hidraulice, pneumatice, electrice, termice sau mecanice, dar sunt acționate din ce în ce mai mult din componente software. Un

dispozitiv de acționare leagă un sistem de control de mediul său [163]. În timp ce senzorii convertesc o schimbare de mediu într-un semnal intern pentru sistem, actuatoarele au o funcție complementară – de a transforma un semnal de control intern în acțiune asupra mediului, Fig. 2.15. Serviciul unui actuator nu se limitează doar la un efect mecanic. Acțiunea asupra mediului se poate face prin temperatură, umiditate, lumină ș.a [164, 165, 166, 167].

Pentru a schimba o stare de mediu, este necesară o acțiune specifică naturii fizice a parametrului de interes al mediului. Natura unei acțiuni este, în majoritatea cazurilor, o natură non-electrică. Deoarece majoritatea echipamentelor contemporane funcționează cu electricitate, este necesară o conversie a energiei electrice în acțiuni non-electrice. Similar cu senzorul, pentru a crea o acțiune, trebuie luate în considerare două componente: controlul convertorului de energie și conversia energiei în acțiune, Fig. 2.15.

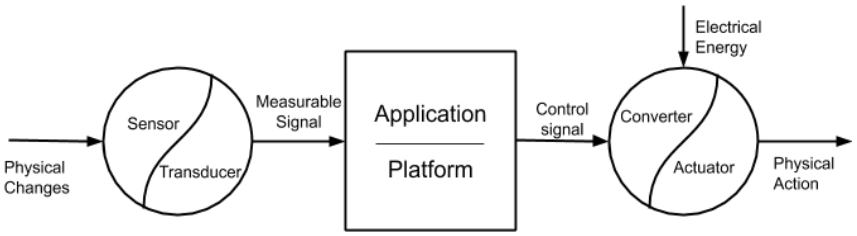


Fig. 2.15. Conceptul generic de Senzor-Actuator [168]

Similar scenariului de condiționare a senzorilor [154, 168] semnalul de acțiune generat urmează o cale de condiționare înainte de a deveni o acțiune asupra mediului. Condiționarea se realizează parțial în domeniul software și parțial în domeniul electric, cum e prezentat în Fig. 2.16.

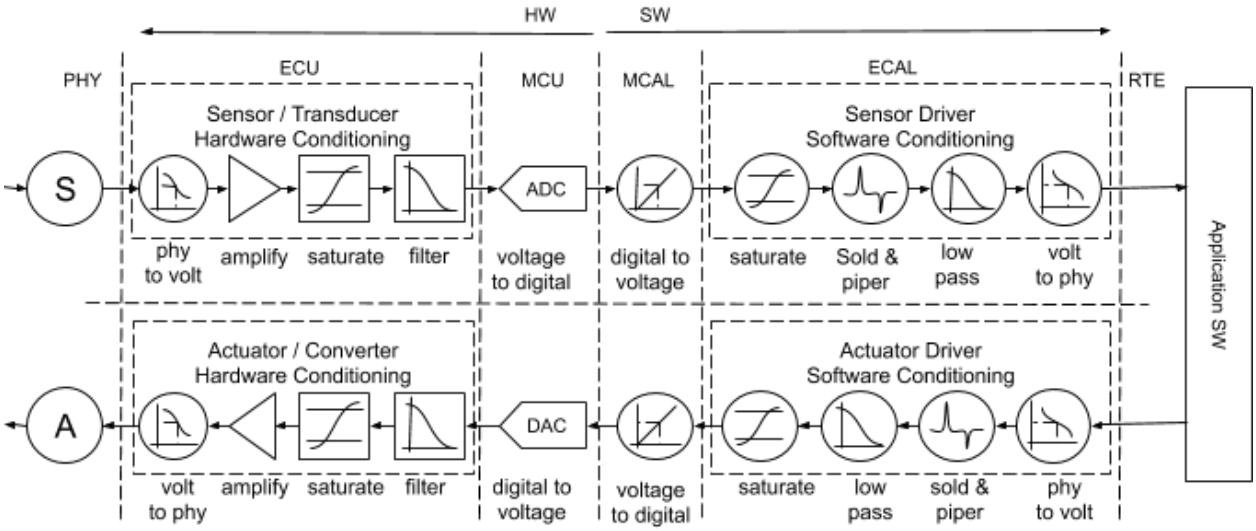


Fig. 2.16. Fluxul de condiționare a semnalului a componentelor Senzor-Actuator [168]

În cazul unui actuator cu o interfață digitală, de exemplu, o soluție integrată, condiționarea poate fi complet implementată în componența actuatorului. La proiectarea actuatorului, condiționarea trebuie făcută, luând în considerare toate aspectele metodei de conversie și mediul

prin care se propagă semnalul înainte de a produce o acțiune fizică. Un exemplu tipic al unei astfel de căi de condiționare cu semnale analogice, în care ordinea funcțiilor depinde de aplicație este prezentat în Fig. 2.16.

Este esențial să se afirme că condiționarea este un *lanț de comunicare* complex și ar trebui luată în considerare atât la nivel de software, cât și la nivel de hardware. În continuare se prezintă descrierea lanțului de condiționare a actuatorului, unde se poate vedea și asemănarea cu lanțul de condiționare a senzorului:

- Conversia software a valorii fizice transformate în valoarea electrică a semnalului aplicat la terminalul electric.
- Filtrarea software ar putea fi aplicată și fluxului de semnal al actuatorului, astfel încât să excludă artefactele nedorite din cauza tranziției interne și a modificărilor intermediare.
- Saturația semnalului software limitează valorile semnalului la minim și maxim. O astfel de operație este necesară pentru a evita generarea de valori ale semnalului electric în afara unui interval valid, acceptate de componentele actuatorului electric.
- Conversia software a valorii semnalului electric într-o configurație digitală pregătește configurația pentru unitatea periferică DAC, care generează semnalul analogic de pe terminalul electric al microcontrolerului.
- Transferul configurației periferice către un semnal electric este etapa în care un semnal software este convertit într-un semnal hardware de către un modul DAC – Convertor Digital Analog.
- Filtrarea electrică este o etapă esențială pentru semnalele analogice, în special pentru semnalele generate de un DAC sau dispozitive cu inerție mică.
- Saturația electrică limitează tensiunea aplicată ca intrare la componentele electronice și vizează protejarea împotriva supratensiunii.
- Aplicarea de putere electrică convertește semnalul de comandă în energie electrică aplicată intrării electrice a actuatorului.
- Actuatorul este convertorul energiei electrice în acțiune asupra mediului fizic.

Semnalul nu este absolut necesar să treacă prin toate fazele menționate, incluse în fluxul tipic de condiționare. Unele dintre ele pot fi omise prin creșterea performanței (de exemplu, reducerea calculelor software), reducerea costurilor (de exemplu, transferul funcționalității de la hardware la software) sau, în general, eliminarea acestora (de exemplu, reducerea filtrelor electrice pentru comutarea semnalelor de control). Uneori se folosesc soluții integrate. Controlul acestora se realizează printr-o interfață de comunicație, iar responsabilitatea pentru asigurarea condiționării este transferată producătorului soluției.

Atât condiționarea hardware, cât și cea software au avantajele și dezavantajele lor, iar acestea sunt discutate în continuare.

Analiza punctelor tari și punctelor slabe ale domeniilor hardware vs. Software

Există cazuri când aceeași operație de condiționare ar putea fi implementată fie în software, fie în hardware. La alegerea celei mai bune metode, este necesar să se analizeze punctele tari și punctele slabe ale acestor metode.

Punctul forte al *condiționării electrice* este operabilitatea, iar partea slabă este faptul că componentele electronice implică costuri. În cazul elementelor de stabilitate și precizie ridicate, aceste costuri au un impact mare asupra producției. Flexibilitatea redusă este un alt dezavantaj. Elementele de reglare implică costuri suplimentare și sunt adesea incomode cu manipularea, precum și fiabilitatea lor. O problemă mult mai mare este schimbarea metodei de condiționare, care implică reorganizarea completă sau înlocuirea întregii componente electronice.

Punctul slab al *condiționării software-ului* este operabilitatea acestuia, presupunând că implică timp de procesare, care este împărțit între mai multe funcționalități ale sistemului. Pe de altă parte, resursele programabile pot fi reprogramate și nu implică costuri materiale. Modificările și ajustările se fac chiar și în timpul funcționării sistemului. Această proprietate are consecința că din ce în ce mai multe funcționalități sunt transferate din domeniul electric în domeniul software, implicând reducerea costurilor de producție ale dispozitivelor.

2.3 Modelare matematică pentru dezvoltarea de componente de sistem a proiectelor

Conform conceptului de *arhitectură generică în straturi* a unui dispozitiv electronic prezentat în Fig. 2.8, sistemele sunt dezvoltate cu componente realizate în domeniul ingineriei software. Aceste componente servesc drept bază pentru realizarea aplicațiilor specifice cerințelor puse fată de sistem. Componentele aplicației se vor abstractiza ca și componente cu intrări, ieșiri și funcții de transfer implementate deseori cu modele matematice. Stratul de aplicații este compus din componente specifice aplicației, Fig. 2.2 și Fig. 2.3. Varietatea modelelor de matematice este una foarte larga cuprinzând și domenii mai netradiționale, cum ar fi sistemele expert cu logică fuzzy [169] sau inteligența artificială. În continuare sunt prezentate câteva modele dezvoltate în scopul obținerii rezultatelor menționate în capitolul 3 din teză.

Modele de evaluare pe bază de interpolare

În contextul sistemelor electronice distribuite dispozitivele implicate în achiziționarea datelor de mediu sunt amplasate la coordonatele fizice specifice ale mediului, oferind informații

colectate despre mediul din zona respectivă. Referindu-ne la ideea de a construi o hartă de mediu în parametrul specific, informațiile colectate definesc doar puncte specifice din zona de interes.

Pentru a obține o continuitate a hărții mediului, se poate aplica o interpolare 2D folosind metoda medie aritmetică ponderată. Adică, fiecare punct nedefinit este calculat pe baza celorlalte puncte definite, ponderea valorii fiecăruia fiind proporțională cu inversul distanței sau apropierea, de acest punct. În acest fel, pe baza valorilor colectate de pe dispozitivele de rețea și metoda liniei de scanare, zonele care nu sunt acoperite sunt restaurate de senzorii vecini, Fig. 2.17.

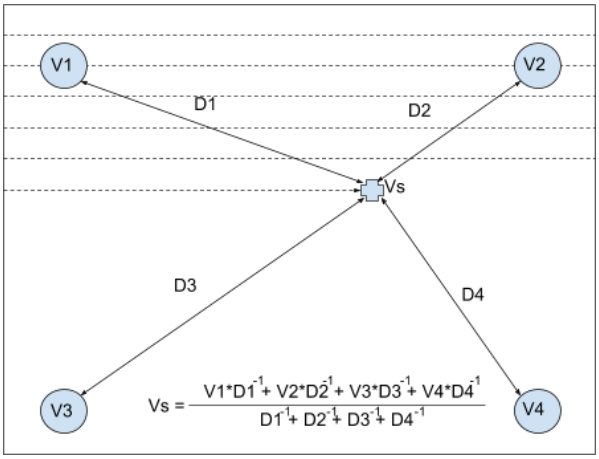


Fig. 2.17. Metoda de interpolare pentru construirea hărții 2D [155]

Valoarea fizică a senzorului virtual se va evalua după principiul mediei aritmetice ponderate, cu ponderile $\frac{1}{D_i}$, după cum urmează în formula (2.3):

$$V_s = \frac{\frac{V_1}{D_1} + \frac{V_2}{D_2} + \frac{V_3}{D_3} + \frac{V_4}{D_4}}{\frac{1}{D_1} + \frac{1}{D_2} + \frac{1}{D_3} + \frac{1}{D_4}} \qquad (2.3)$$

unde V_s este valoarea evaluată pentru senzorul virtual, V_i – valoarea colectată de la senzorii fizici, D_i – distanța dintre punctul de referință a senzorului fizic și punctul de referință a senzorului virtual. Mărimea $\frac{1}{D_i}$, inversul distanței, este considerată ca mărime de apropiere.

Abordarea dată este o metodă relativ simplă pentru obținerea unei hărți de o precizie satisfăcătoare. Pentru o precizie mai mare, ar fi necesar să se mărească numărul de dispozitive de achiziție, dar și alegerea unei metode mai sofisticate de calcul de interpolare, care ar crește cerințele de performanță ale dispozitivului de procesare a datelor și ar afișa harta.

Model de control prin metoda ON-OFF cu histerezis

Cel mai simplu model de control al parametrilor fizici la o valoare prestabilită este controlul prin metoda ON-OFF. Acesta presupune activarea sau dezactivarea componentelor de acționare asupra parametrului fizic de interes ca rezultat al comparației acestuia cu o valoare

prestabilită dorită. O variație mai eficientă a acestei metode este control *ON-OFF cu histerezis*, care presupune menținerea valorii curente a parametrului fizic controlat între două valori de minimum și maxim, ceea ce permite relaxarea sistemului și reduce numărul de comutații al cheilor electronice de putere.

Modelul de control *ON-OFF cu histerezis* este realizat cu o funcție de transfer, după cum urmează în formula (2.4):

$$R(t+\Delta t) = \begin{cases} K_C, \; dacă \; V_{act}(t) < V_{on}(t) \\ 0, \; dacă \; V_{act}(t) > V_{off}(t) \\ R(t), \; altfel \end{cases} \quad (2.4),$$

unde $R(t)$ este valoarea curentă de ieșire pentru componenta de acționare; $R(t+\Delta t)$ – valoarea estimată următoare; K_C – valoarea nominală de control pentru sistemul de acționare în starea ON; $V_{act}(t)$ – valoarea actuală a parametrului fizic; $V_{on}(t)$ – valoarea pragului de activare a componentei de acționare; $V_{off}(t)$ – valoarea pragului de dezactivare a componentei de acționare.

Conform legității de histerezis, evoluția și revenirea la inițial a valorii de acționare se produce pe trasee diferite, după cum urmează în Fig. 2.18.

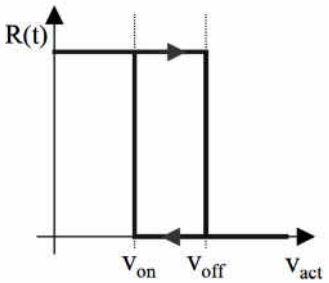


Fig. 2.18. Evoluția valorii de acționare, conform legității de histerezis [171]

Evoluția valorii actuale a parametrului V_{act} va varia între valorile V_{on} și V_{off}, relaxând astfel solicitarea în exces de comutații în sistemul de acționare al sursei de temperatură, vezi Fig. 2.19 pentru o reprezentare grafică a celor expuse.

Această metodă de control este utilizată pe larg în sisteme cu o inerție mare, unde procesele de fluctuații în parametrul fizic de interes sunt relativ mici și nu răspund imediat la implicările de la componentele de acționare.

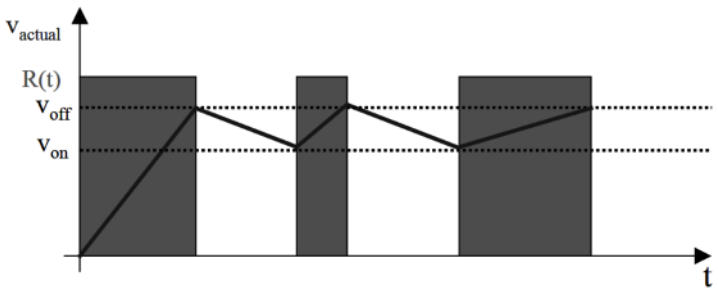


Fig. 2.19. Evoluția valorii temperaturii după principiul ON-OFF cu histerezis [171]

Model de control prin metoda Proporțional Integral Diferențială

Pentru sisteme cu o inerție mai mică, care necesită un control mai fin, una din metodele utilizate pe larg este metoda *Proporțional Integral Diferențială* (PID) [171]. În cazul acestei metode valoarea de ieșire a componentelor de acționare este evaluată în baza valorii de deviație de la valoarea dorită V_{des} , numită eroare. Calculul deviației valorii curente V_{act} a parametrului controlat de la valoarea dorită pentru control, se va calcula conform formulei (2.5):

$$e = (V_{des} - V_{act}) \tag{2.5}$$

Calculul componentei proporționale, unde K_P coeficientul pentru componenta *proporțională* se va realiza conform formulei (2.6):

$$R_P(t) = K_P \cdot e(t) \tag{2.6}$$

Calculul componentei diferențiale, unde K_D coeficientul pentru componenta *diferențială* se va realiza conform formulei (2.7):

$$R_D(t) = K_D \cdot \frac{de}{dt} = K_D \cdot \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{2.7}$$

Calculul componentei integrale, unde K_I coeficientul pentru componenta *integrală* se va realiza conform formulei (2.8):

$$R_I(t) = K_I \cdot \int_0^t e(t)dt = K_I \cdot \sum_{k=0}^{n} e_k \cdot \Delta t \tag{2.8}$$

Calculul valorii de control ca sumă a celor trei componente ale modelului de control se va realiza conform formulei (2.9):

$$R_{PID}(t) = R_P(t) + R_I(t) + R_D(t) = K_P \cdot e(t) + K_I \cdot \sum_{k=0}^{n} e_k \cdot \Delta t + K_D \cdot \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{2.9}$$

Pentru adaptarea coeficienților K_P , K_I și K_D s-a utilizat o metodă iterativă [171], care constă în următorii pași:

- setare V_{des} dorită , $K_P = 0$, $K_I = 0$, $K_D = 0$;
- mărire K_P până la oscilare, împărțire Kp/2;
- mărire K_D pâna se observă o creștere cu 5-10%:
- mărire K_I până la oscilare, împărțire $\frac{K_I}{2}$ sau $\frac{K_I}{3}$;
- verificare cu diverse valori V_{des}.

Modele matematice pentru componente control mecatronică

Pentru sisteme cu aplicații în mecatronică, dezvoltarea componentelor presupune dezvoltarea de modele matematice și proiectarea mecanismelor de cinematică pentru

componentele mecanice care le compun. Pentru sistemul de control al unui braț robotic este necesară modelarea cinematicii directe și inverse.

Cinematica inversă pentru un braț robotic cu trei grade de libertate, 3 DOF, cu două articulații este evaluată folosind calcule geometrice, vezi Fig. 2.20.

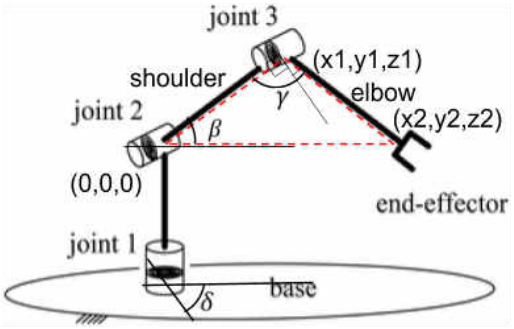


Fig. 2.20. Modelul brațului 3-DOF [168]

Proiecția r_{xy} al distanței pe planul O_{xy} se va calcula conform formulei (2.10):

$$r_{xy} = \sqrt{x^2 + y^2} \tag{2.10}$$

unde x și y sunt coordonatele capului brațului robotizat pe plan orizontal.

Distanța până la efectorul final al brațului r_{xyz} se calculează conform formulei (2.11):

$$r_{xyz} = \sqrt{r_{xy}^2 + z^2} \tag{2.11}$$

unde z este coordonata verticală a capului.

Iar unghiul de rotație al bazei δ poate fi determinat după formula (2.12):

$$\delta = \operatorname{asin}\frac{x}{r_{xy}} \tag{2.12}$$

Pentru unghiul umărului β vom avea formula (2.13):

$$\beta = \operatorname{asin}\frac{z}{r_{xyz}} + \operatorname{acos}\frac{l_1^2 + r_{xyz}^2 - l_2^2}{2 * l_1 * |r_{xyz}|} \tag{2.13}$$

unde l_1 și l_2 sunt lungimea segmentelor de braț.

Pentru unghiul cotului brațului γ vom avea formula (2.14):

$$\gamma = \operatorname{acos}\frac{l_1^2 + l_2^2 - r_{xyz}^2}{2 * l_2 * l_1} \tag{2.14}$$

Pentru modelul de calcul al cinematicii directe a brațului se aplică algebra vectorială, conform formulei (2.15) pentru coordonata z a capului efectorului brațului robotic:

$$z = l_1 * \sin(\beta) + l_2 * \sin\left(\beta + \gamma - {}^{\pi}/_{2}\right) \tag{2.15}$$

Formula (2.16) pentru distanța pe planul O_{xy}:

$$r_{xy_2} = l_1 * \cos(\beta) + l_2 * \cos\left(\beta + \gamma - {}^{\pi}/_{2}\right) \tag{2.16}$$

Formula (2.17) pentru coordonata x a capului efectorului brațului robotic:

$$x_2 = r_{xy_2} * \sin(\delta) \quad (2.17)$$

Și formula (2.18) pentru coordonata y a capului efectorului brațului robotic:

$$y_2 = r_{xy_2} * \cos(\delta) \quad (2.18)$$

unde x_2, y_2, z_2 sunt coordonatele carteziene pentru cap și r_{xy_2} – proiecția distanței până la cap pe plan orizontal de la originea brațului (0,0,0), vezi Fig. 2.20.

Model de control comportamental cu Automate Finite

Pentru definirea comportamentului unui sistem deseori este recomandat controlul cu Automate Finite (AF). Controlul cu AF reprezintă o abstracție ce descrie o soluție a unei probleme care definește comportamentul sistemului sub formă de mecanism, care își schimba stările ca reacție la intrările sistemului și produce ieșiri corespunzătoare.

Modelul automatului finit este reprezentat prin funcții caracteristice pentru definiția unui automat finit, cum ar fi evaluarea intrărilor, evaluarea următoarei stări a AF și evaluarea ieșirilor. Pentru evaluarea intrărilor, unde $in_1, in_1 \dots in_n$ sunt totalitatea de intrări ale sistemului, se folosește funcția (2.19):

$$Input = f(in_1, in_1 \dots in_n) \quad (2.19)$$

Pentru evaluarea următoarei stări $NextState$ a automatului finit se va utiliza expresia conform formulei (2.20):

$$NextState = f(Input, CurrentState) \quad (2.20)$$

Pentru evaluarea ieșirilor se folosește funcția (2.21):

$$Output = f(CurrentState) \quad (2.21)$$

diagrama de stări a Automatului finit va arăta după cum este prezentat în Fig. 2.21.

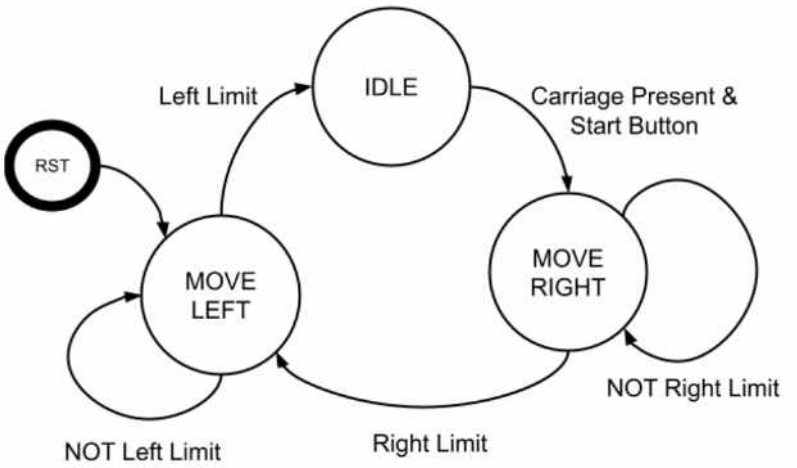


Fig. 2.21. Diagrama Automatului Finit pentru un mecanism de încărcare

Conform diagramei de stări a AF de control, componentele acestuia sunt reprezentate de:

- Stări ale automatului finit: IDLE – stare de așteptare, MOVE RIGHT – stare de împingere a cărucioarelor încărcate, și MOVE LEFT – stare de revenire a tijei în starea inițială. MOVE LEFT este starea inițială a automatului.
- Intrări: Left Limit și Right Limit sunt limitatoarele de cursă a tijei de încărcare. Start Button este butonul de declanșare-încărcare. Carriage Present este detectorul de prezență a căruciorului la intrare în camera de uscare.
- Ieșiri: Go Left reprezintă semnalul de pornire a motorului pentru a reveni în starea inițială. Go Right reprezintă semnalul de pornire a motorului pentru împingerea căruciorului cu fructe în interiorul camerei de uscare.

Una din metodologiile de a defini funcțiile de transfer ale AF pentru evaluarea intrărilor, ieșirilor și a tranzițiilor este prin reprezentare tabelară, Tabelul 2.1, care reprezintă asocieri între intrările și ieșirile acestor funcții de transfer, și pot fi implementate prin tabele veridice sau metode asociative similare.

Tabelul 2.1. Evaluarea stărilor și ieșirilor AF a mecanismului de încărcare

Num	Current State	Output {Go Left, Go Right}	State Delay	Input {LeftLimit, RightLimit, StartButon & CarriagePresent]				
				00xb	01xb	100b	101b	11xb
0	IDLE	00b	100 ms	IDLE	IDLE	IDLE	MOVE RIGHT	IDLE
1	MOVE RIGHT	01b	100 ms	MOVE RIGHT	MOVE LEFT	MOVE RIGHT	MOVE RIGHT	MOVE RIGHT
2	MOVE LEFT	10b	100 ms	MOVE LEFT	MOVE LEFT	IDLE	IDLE	IDLE

2.4 Metode de diagnoză și protecție în sistemele electronice distribuite

Serviciile pentru senzor sunt furnizate de SW din stratul superior al componentei senzorului. Acest strat este furnizorul de servicii către aplicație și asigură funcționalitățile de gestionare a componentei, precum și interacțiunea cu celelalte servicii din alte componente în cadrul RTE. Funcțiile prefixate de *Get, Scan, Read, Recv* ale componentelor, sunt destinate pentru a aduce informații din mediu extern către straturile superioare. La acest nivel superior, sub RTE, este locul potrivit pentru managerul de senzori, care permite construirea de caracteristici speciale ale senzorilor pe baza mai multor surse de informații. Un bun exemplu ar putea fi implementarea senzorilor virtuali, care furnizează informații fără a colecta informații de la un senzor real, ci printr-un model special.

În stratul de servicii se implementează detectarea simptomelor specifice senzorului și calificarea diagnosticului, oferind aceste informații întregului sistem. S-ar putea numi câteva simptome de diagnostic generice pentru senzori, la fel ca metodele de calificare și tipul de reacții asociate, care sunt prezentate în subsecțiunile de mai jos.

Metode de diagnosticare a simptomelor în cadrul semnalului

Diagnosticul este identificarea naturii și a cauzei unui anumit fenomen. Diagnosticul este utilizat în multe discipline diferite, cu variații de utilizare a logicii, analizei și experienței, definind „modelul cauză / efect" adecvat. În ingineria sistemelor și informatică este utilizat, de regulă, pentru a determina cauzele simptomelor, atenuărilor și soluțiilor [172]. Simptomul diagnosticului vine ca un indicator de detectare a situației în dimensiunea logică „0", „1" sau LOW, HIGH. Ca diagnostice primare asupra semnalelor primite de la senzori se regăsesc următoarele:

Simptom de prag – implică compararea unui semnal cu o valoare prestabilită numită prag. În cazul în care valoarea semnalului este mai mare decât o presetare, se identifică simptomul de supratensiune (de exemplu, supratensiune, supratemperatură), iar în cazul în care este mai mică decât valoarea de referință, se identific ă simptomul valorii scăzute (de exemplu, sub tensiune, sub temperatură), Fig. 2.22.

Calculul simptomului de supra nivel se va evalua conform formulei (2.22):

$$OverTh(t) = \begin{cases} 1, & PHY(t) \geq treshold \\ 0, & PHY(t) < treshold \end{cases} \quad (2.22)$$

unde $OverTh$ este simptomul de supra nivel, PHY este valoarea fizică a parametrului monitorizat, iar $treshold$ – valoarea de referință pentru evaluarea simptomului.

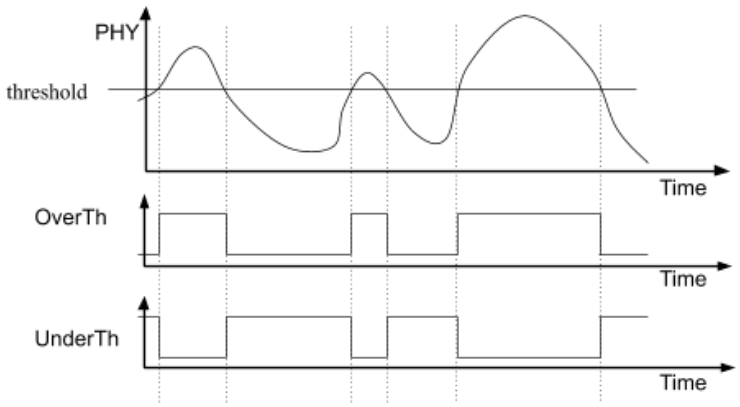


Fig. 2.22. Detectare simptom de prag [154]

Calculul simptomului de sub nivel $UnderTh$ se va evalua conform (2.23):

$$UnderTh(t) = \begin{cases} 0, & PHY(t) \geq treshold \\ 1, & PHY(t) < treshold \end{cases} \quad (2.23)$$

Simptomul de interval – presupune poziția valorii măsurate între două valori presetate. În acest caz se pot identifica patru simptome: în interval, deasupra intervalului, sub interval, în afara intervalului, Fig. 2.23.

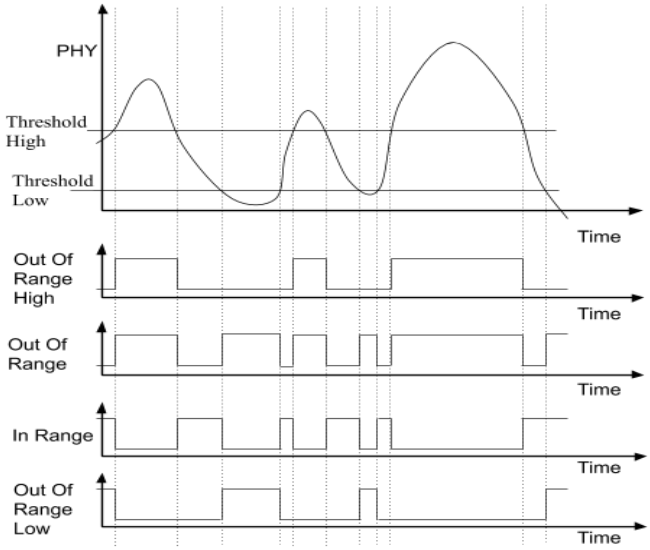


Fig. 2.23. Detectare simptom de interval [154]

Simptomul de supra nivelului de sus al intervalului se va evalua conform (2.24):

$$OORH(t) = \begin{cases} 1, & PHY(t) \geq treshold\ high \\ 0, & PHY(t) < treshold\ low \end{cases} \quad (2.24)$$

Simptomul din afara intervalului se va evalua conform (2.25):

$$OOR(t) = \begin{cases} 1, & PHY(t) \leq treshold\ low \\ 1, & PHY(t) \geq treshold\ high \\ 0, & treshold\ high > PHY(t) > treshold\ low \end{cases} \quad (2.25)$$

Simptomul încadrare în interval se va evalua conform (2.26):

$$IR(t) = \begin{cases} 1, & PHY(t) \leq treshold\ low \\ 1, & PHY(t) \geq treshold\ high \\ 0, & treshold\ high > PHY(t) > treshold\ low \end{cases} \quad (2.26)$$

Simptomul de sub nivelul de jos al intervalului se va evalua conform (2.27):

$$OORL(t) = \begin{cases} 1, & PHY(t) \geq treshold\ high \\ 0, & PHY(t) < treshold\ low \end{cases} \quad (2.27)$$

Simptomul de plauzibilitate – verifică plauzibilitatea măsurătorilor pe baza dobândirii aceluiași parametru din două sau mai multe surse. Dacă diferența de semnal este mai mare decât limita admisă, va fi generat un simptom de eroare de plauzibilitate, Fig. 2.24.

Simptomul de eroare de plauzibilitate se va evalua conform (2.28):

$$PlausFail(t) = \begin{cases} 1,\ PHY_1(t) - PHY_2(t)\ \leq treshold\ low \\ 1,\ PHY_1(t) - PHY_2(t) \geq treshold\ high \\ 0, treshold\ high > PHY_1(t) - PHY_2(t) > treshold\ low \end{cases} \quad (2.28)$$

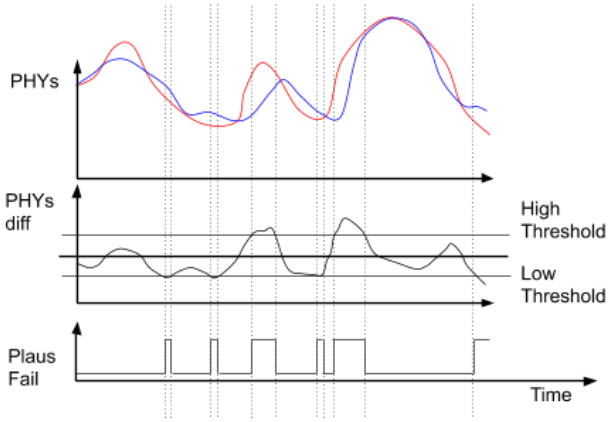


Fig. 2.24. Detectare simptom plauzibilitate [154]

Simptom blocare în interval. Indiferent de natura semnalului, există întotdeauna unele variații ale datelor obținute. Acest simptom detectează înghețarea evoluției semnalului, care ar putea fi cauzată de o disfuncționalitate a sursei de semnal, cum ar fi, de exemplu, un scurtcircuit sau o deteriorare a conexiunii, Fig. 2.25.

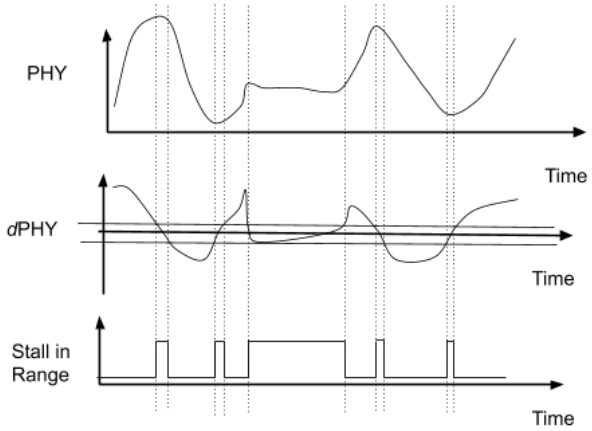


Fig. 2.25. Detectare simptom blocare în interval [154]

Simptomul de blocare în interval se va evalua conform (2.29):

$$IR(t) = \begin{cases} 1, \dfrac{dPHY}{dt}(t) \leq treshold\ low \\ 1, \dfrac{dPHY}{dt}(t) \geq treshold\ high \\ 0, treshold\ high > \dfrac{dPHY}{dt}(t) > treshold\ low \end{cases} \quad (2.29)$$

Metode de calificare a diagnozelor

Pentru ca orice simptom să fie considerat diagnostic valid și să fie procesat de sistemele sensibile la acest diagnostic, este necesară o prelucrare preliminară a simptomelor pentru a le califica în diagnoze. Calificarea diagnosticelor ar putea fi implementată printr-un filtru binar, care presupune că un diagnostic poate fi calificat sau descalificat dacă simptomele persistă mai mult

decât o anumită perioadă. Filtrul binar în acest context filtrează un semnal binar și ar putea fi implementat cu un contor anti-bouncing, Fig. 2.26.

Calificare: Când apare simptomul, contorul anti-bouncing va crește cu o valoare prestabilită ABC_INC. La atingerea unei valori ABC_MAX, diagnosticul va fi considerat calificat și va păstra starea activă până la descalificarea acestuia. În caz că diagnoza este calificată, iar simptomul încă mai este valid – contorul este saturat la o valoare maximă – ABC_MAX.

Descalificare: Când simptomul dispare, contorul va scădea cu o valoare presetată ABC_DEC. La atingerea unei valori ABC_MIN, diagnosticul va fi considerat descalificat și va rămâne în această stare până la următoarea calificare. Contorul în cazul absenței unui simptom de diagnostic este saturat la valoarea minimă – ABC_MIN.

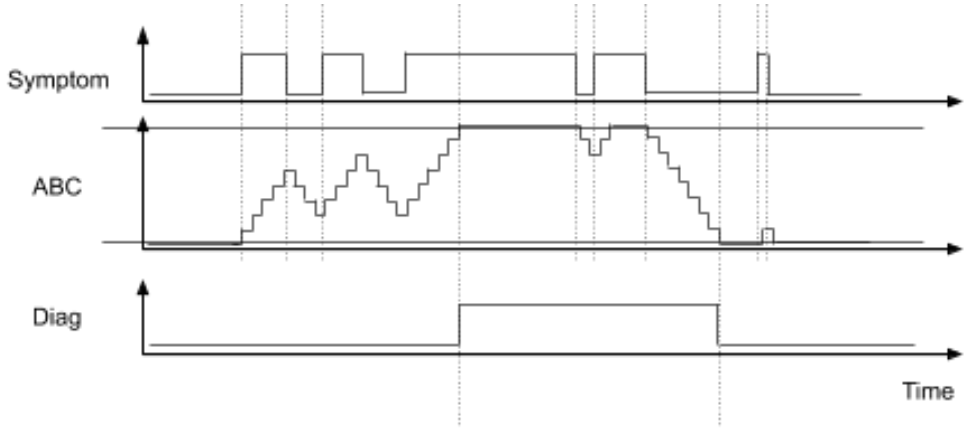


Fig. 2.26. Calificare diagnoze cu contor anti-bounce [154]

Contorul de calificare a unei diagnoze se va evalua conform (2.30):

$$ABC(t+\Delta t)=\begin{cases} ABC_{MAX}, & ABC(t)\geq ABC_{MAX} \\ ABC(t)+1, & Simptom(t)=1 \text{ și } ABC_{MIN}<ABC(t)<ABC_{MAX} \\ ABC(t)-1, & Simptom(t)=0 \text{ și } ABC_{MIN}<ABC(t)<ABC_{MAX} \\ ABC_{MIN}, & ABC(t)\leq ABC_{MIN} \end{cases} \quad (2.30)$$

iar calificarea diagnozei conform (2.31):

$$Diag(t+\Delta t)=\begin{cases} 1, & ABC(t)\geq ABC_{MAX} \\ 0, & ABC(t)\leq ABC_{MIN} \\ Diag(t), & ABC_{MIN}<ABC(t)<ABC_{MAX} \end{cases} \quad (2.31)$$

Metode de reacție la diagnosticarea calificată

Odată ce un diagnostic este calificat, acesta trebuie asociat cu o reacție specifică. Reacțiile obișnuite ar putea fi enumerate, după cum urmează:

- *Inhibare* – presupune ignorarea simptomului detectat. De exemplu, a apărut o situație în care diagnosticul nu este relevant sau persistă o altă situație cu prioritate superioară.

- *Informare* – implică transmiterea unui mesaj către o interfață cu utilizatorul în scop informativ, de exemplu, o recomandare de încărcare a bateriei dispozitivului.
- *Blocarea* – implică dezactivarea anumitor funcționalități în timpul unui diagnostic valid, de exemplu, blocarea unei acțiuni mecanice în timpul prezenței unei persoane în zona de pericol.
- *Deratarea* – implică limitarea funcționalităților la detectarea unor diagnostice specifice, de exemplu, limitarea consumului de energie la detectarea unei baterii cu nivel scăzut de încărcare sau limitarea luminozității ecranului laptopului. Deratarea poate fi una binară sau poate urma o funcție de deratare urmând o funcție MIN între puterea aplicată și funcția de deratare în funcție de semnalul de intrare.

Comparații diagnoze interne vs externe

În funcție de sursa de semnal pentru diagnostice, pot fi identificate anumite situații specifice și se pot efectua reacțiile adecvate, Fig. 2.27.

Diagnosticul intern sau autodiagnosticul verifică simptomele interne ale funcționării sistemului pentru a se asigura că sistemul se comportă în limite admisibile și că nu apare nicio defecțiune a sistemului în timpul funcționării dispozitivului. Detectarea diagnosticului intern poate preveni la timp defecțiunile sistemului care implică reacția adecvată înainte de orice deteriorare a dispozitivului. De exemplu, dacă diagnosticul în interval (IR), conform (2.26), va fi aplicat pe un semnal inițial necondiționat (RAW), înainte de a efectua orice condiționare, evaluarea va detecta că senzorul furnizează niveluri de tensiune care sunt în afara limitelor, în concluzie va rezulta că senzorul este defect sau este o problemă de conectare, electrică, fie scurt la pământ sau la sursă de alimentare, fie fir rupt. În acest caz, reacția ar putea fi deconectarea funcționalităților dependente de sursa de semnal respectivă.

Diagnosticul extern este realizat pentru detectarea simptomelor din mediul exterior și utilizat în mecanismele de comportament funcțional al sistemului. Pentru exemplul de evaluare al diagnosticul pe o valoare fizică, punctul de diagnostic este după întreg fluxul de condiționare. În cazul în care se detectează OOR (în afara intervalului), iar diagnosticul de defecte ale senzorului, (diagnostic intern) nu raportează erori, concluzia poate fi că intervalul de temperatură ambiantă este în afara normele admise și reacția poate fi un mesaj de recomandare, o reducere a acțiunii de încălzire sau activare răcire, asupra mediului pentru a evita evoluția situației. Un exemplu mai concret ar fi supraîncălzirea motorului la turații mari.

La fel ca condiționarea, diagnosticele și reacțiile pot fi implementate în ambele domenii, în domeniul electronic și în domeniul software. Metodele de implementare a mecanismelor sunt specifice domeniului.

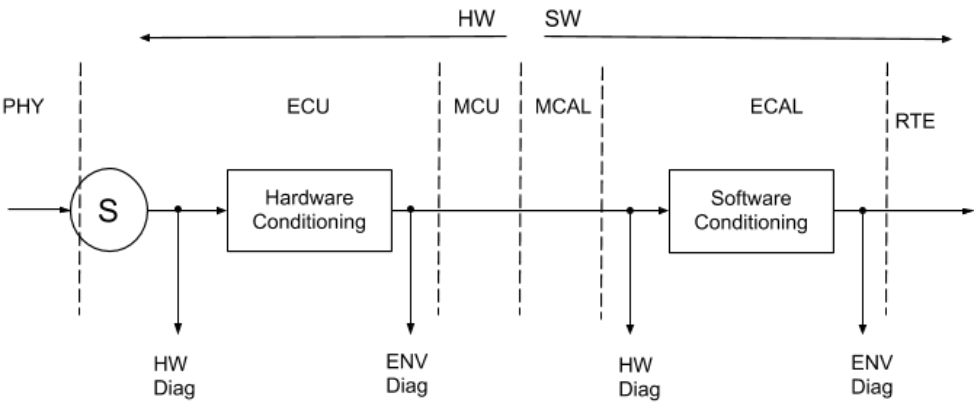


Fig. 2.27. Exemplu sursă de diagnoze [154]

Metode de protecții pentru componentele de acționare

Diagnosticul este identificarea naturii și a cauzei unui anumit fenomen. Diagnosticul este utilizat în multe discipline diferite, cu utilizarea variată a logicii, analizei și experienței, definind modelul adecvat de cauză / efect. În ingineria sistemelor și informatică, este de obicei utilizat pentru a determina cauzele simptomelor, atenuărilor și soluțiilor [154, 172]. Diagnosticul este legat în principal de evaluarea simptomelor pentru diferite surse de semnal și este de obicei asociat cu fluxul de achiziție a semnalului. În referința [154], autorii discută stiva componentelor de achiziție a senzorilor, unde pragul, intervalul, plauzibilitatea sunt unele dintre cele mai frecvente simptome asociate senzorilor. Prin urmare, componentele de diagnostic ar putea fi clasificate ca componente auxiliare. Scopul principal al componentelor auxiliare este monitorizarea funcțiilor sistemului, colectarea informațiilor și raportarea problemelor. Similar cu stiva senzorului și diagnosticul, actuatoarele sunt asociate cu componente specifice pentru protecție. Scopul acestor componente este de a proteja actuatorul, dispozitivul și mediul ca reacție la simptomele de diagnostic, stările sistemului sau prevenirea situațiilor nedorite prin ajustarea comportamentului actuatorului. O propunere de proiectare pentru asocierea senzor-actuator cu componentele de protecție a diagnosticului este prezentată în Fig. 2.28.

Pe lângă diagnosticare, protecția este implementată în sistemele electrice și software. Protecția aplicată semnalului înainte de condiționare este legată de protecția funcțională și vizează acțiunea aplicată manipulării mediului. Protecția aplicată după condiționare vizează protecția internă a sistemului. Diferite tipuri de protecție ar putea fi aplicate unui actuator, dar numai câteva sunt tratate în teză. Cea mai simplă metodă de protecție este prin saturarea valorii semnalului la o cantitate maximă pentru curent, tensiune sau putere. Saturația valorii semnalului se aplică protecției la suprasarcina electrică a sistemului.

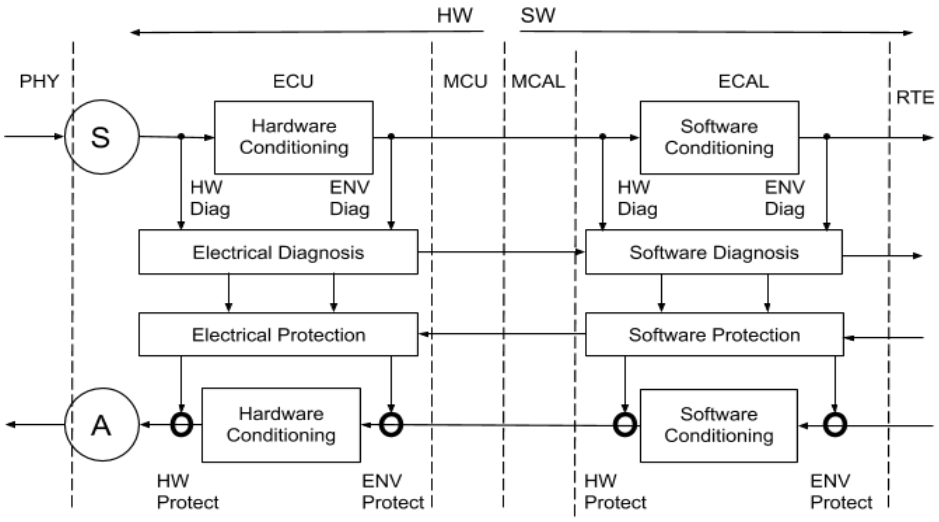


Fig. 2.28. Asocierea componentelor senzor-actuator cu diagnoză-protecție [168]

Protecția prin deratare este o metodă de protecție dinamică a saturației, unde valoarea maximă depinde de un alt semnal de sistem. Un exemplu este reducerea termică a curentului de ieșire, așa cum se arată în Fig. 2.29 și evaluată conform (2.32):

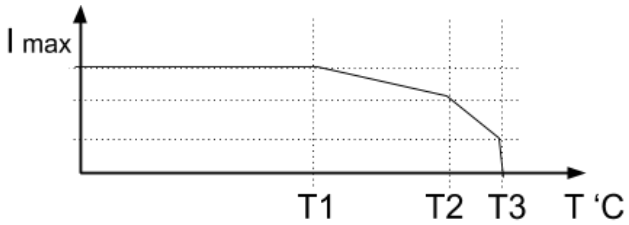


Fig. 2.29. Exemplu de derata a curentului maxim în funcție de temperatură [168]

$$I_{out}(t) = \begin{cases} I_{max}, & T \le T_1 \\ \dfrac{(T_1 - T_2)\,(I_{max} - I_1)}{T - T_1} + I_{max}, & T_1 \le T \le T_2 \\ \dfrac{(T_1 - T_2)\,(I_1 - I_2)}{T - T_1} + I_1, & T_2 \le T \le T_3 \\ 0\,, & T \ge T_3 \end{cases} \qquad (2.32)$$

unde I_{out} este valoarea limitării de curent după evaluarea deratării, I_{max} curentul maxim admisibil, I_1 , I_2 – punctele critice în curent pentru funcția de derata, T – Temperatura curentă măsurată, T_1 , T_2, T_3 – punctele critice în temperatură pentru funcția de derata, conform Fig. 2.29.

Protecția prin controlul evoluției traiectoriei parametrilor, cum ar fi controlul mișcării pentru un control numeric computerizat (CNC) sau braț robotizat constă în îmbunătățirea comportamentului general al mașinii și reducerea erorii de urmărire a traseului. Referințele profilului trebuie proiectate luând în considerare nu numai poziția finală, ci și valorile de vârf ale vitezei, accelerației și impulsului. Jerk este un parametru dinamic esențial, deoarece generează vibrații și produce erori de traiectorie excesive în mișcarea de mare viteză [173, 174], Fig. 2.30.

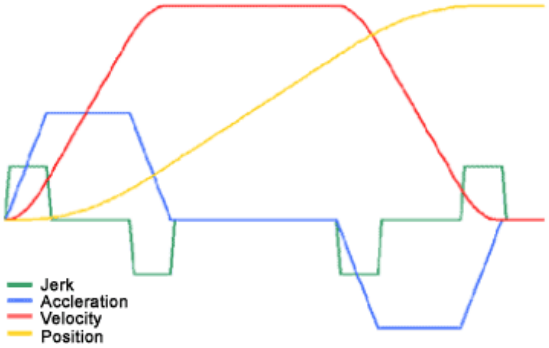


Fig. 2.30. Exemplu de evoluție a traiectoriei [175]

Super accelerația $j(t)$ – jerk, reprezintă derivata accelerației în timp și se va evalua conform expresiei (2.33):

$$j(t) = \frac{da(t)}{dt} = \frac{d^2v(t)}{dt} = \frac{d^3r(t)}{dt} \tag{2.33}$$

unde a este accelerația (m/s^2), v – viteza (m/s), r – deplasare (m).

Protecția prin Automat Finit permite implementarea unei metode de protecție mai sofisticate. Un exemplu este protecția unui motor sub control pornit-oprit prin limitarea timpului activ, asigurând un timp mort între stările active. Vezi un exemplu în Fig. 2.31.

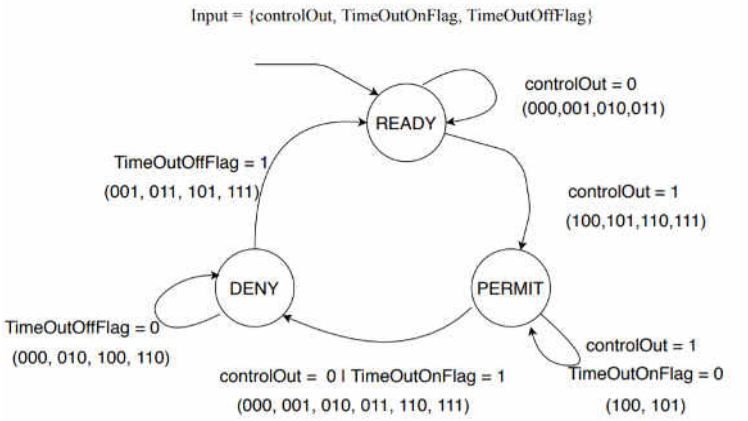


Fig. 2.31. Exemplu protecție motor cu Automat Finit [168]

2.5 Comunicare între componente

2.5.1 Metode de clasificare a procesului de comunicare

Pentru un sistem IoT ca dispozitiv electronic extins, vom clasifica procesul de comunicare care conduce informațiile prin întregul sistem care conectează componentele sale. În lucrarea [156] sunt prezentate două clasificatoare primare: interacțiunea și domeniul. Fiecare dintre acestea descrie o perspectivă și definește aspectele clasificatorului.

Clasificatorul de interacțiuni – se referă la interacțiunile din întregul sistem și constă din două tipuri de interacțiuni cu dispozitivele și cu mediul.

- Interacțiunea dispozitiv-dispozitiv reprezintă toată infrastructura de rețea care interconectează părțile dispozitivului electronic extins. În referințele clasice, se referă la comunicarea inter-dispozitiv. O abstractizare pentru acest clasificator este prezentată în Fig. 2.5. Acest grup include toate tehnicile și metodele de conectivitate, începând de la legătura cu fir cu rază scurtă de acțiune printr-un protocol SPI sau I2C până la cele mai complexe, cum ar fi wireless prin 4G sau LoRa, ca legătură fizică și stiva TCP / IP pentru abstractizarea software-ului. Pentru conceptul de dispozitiv electronic extins, mecanismele de comunicație dispozitiv-dispozitiv obstrucționează interconectarea electrică între părțile dispozitivului extins, oferind straturile superioare de abstractizare ale aplicației pentru a interacționa cu sistemul ca un tot întreg.
- Interacțiunea dispozitiv-mediu reprezintă o abstracție a întregii interfețe a sistemului cu mediul fizic, prezentată conceptual în Fig. 2.32. Prin conceptul sistemului de dispozitiv electronic distribuit, toate componentele senzorului și actuatorului din rețeaua sa sunt accesibile către straturile superioare ale aplicației ca și cum fiind situate pe același dispozitiv fizic [168]. Interfața om-mașină (Human Machine Interface – HMI) este considerată un caz particular al componentelor senzor-actuator proiectate pentru interacțiunea cu utilizatorul, astfel încât acestea sunt, de asemenea, clasificate ca interacțiune dispozitiv-mediu.

Fig. 2.32. Interacțiuni dispozitiv cu mediul [156]

Clasificator de domeniu – clasifică interconectarea comunicației după afilierea domeniului. Această interacțiune de comunicare clasificator ar putea fi afiliată domeniului hardware (HW) sau domeniului software (SW).

- Domeniul hardware constă din componente electronice interconectate, cum ar fi unele părți mecanice implicate în fluxul de semnal între interacțiunea mediului fizic, cum ar fi senzorul sau mediul fizic al procesului de comunicare și domeniul software. Stiva de acționare a senzorilor constă din toate caracteristicile incluse în domeniul hardware, cum ar fi transducția, conversia puterii și condiționarea hardware. Stivele de comunicare sunt compuse din cipurile de comunicare și caracteristicile de gestionare a protocoalelor hardware. Canalul de comunicare fizic în sine, de exemplu, perechea răsucită de cupru sau magistrala I2C trasă, se referă la domeniul hardware.

- Domeniul software constă din resursele software care constau din toate metodele SW aplicate canalelor de informații care interconectează toate componentele aplicației și conduc semnalul prin componentele aplicației. Senzorul, actuatorul, interacțiunea cu utilizatorul și stivele SW de comunicare sunt tratate în mod similar, ca flux de informații cu lanțuri de funcții de transfer pentru transferul de informații. În acest domeniu, mecanismele de comunicare inter-proces [176] și interfețele de componente [177] sunt implicate în transferul de informații între componentele aplicației.

2.5.2 Modelarea componentelor de comunicare și modelarea lanțului de comunicare în sistemele electronice distribuite

Comunicarea se referă la procesul de transfer al informațiilor de la sursă la destinație printr-un canal de comunicare. Această afirmație nu se limitează la domenii specifice. O mulțime de asemănări ar putea fi evidențiate în comunicarea interpersonală și comunicarea IT. În ambele, vom avea componente similare ale comunicării, cum ar fi sursa și expeditorul, canalul, receptorul și destinația, mesajul, feedback-ul, codificarea, decodarea și zgomotul sau barierele, Fig. 2.33.

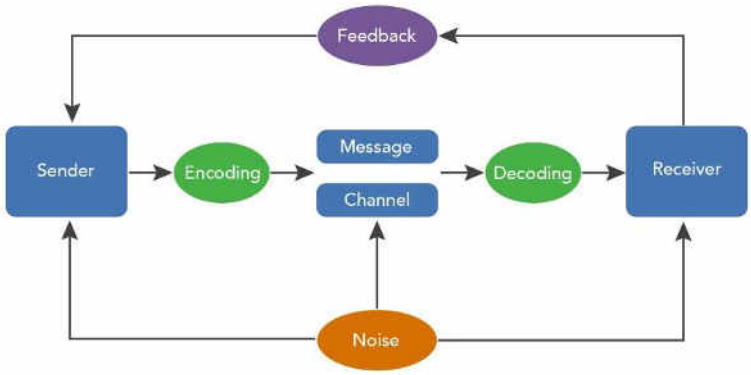


Fig. 2.33. Componentele cheie ale procesului de comunicare [178]

Urmând conceptul de interacțiune dispozitiv-mediu din Fig. 2.32, sursa inițială de informații pentru un dispozitiv este mediul fizic, achiziționată de o componentă de detectare și transferată aplicației, unde o reacție este generată și aplicată înapoi către mediul fizic. Conceptul este, de asemenea, utilizat pentru sursa de informații a rețelei fără fir, având în vedere că semnalul radio ar trebui să fie detectat înainte de a se converti în fluxul de informații. Un bun exemplu care justifică conceptul este comunicarea morse audio sau prin iluminare. Înainte de a construi un cadru de caractere, punctul și liniuța ar trebui să fie detectate mai întâi în semnalul sonor sau luminos, urmând fluxul clasic al semnalului de condiționare a senzorului.

Luând în considerare tezele de mai sus, *lanțul de comunicare* pentru un dispozitiv IoT va fi constituit din componentele incluse în lanțul de acționare prin detectare combinate cu componentele pentru transferul sau transportul de informații. Prin această ipoteză, colectarea

completă a informației prin intermediul componentelor privind procesul de comunicare constă din lanțul de componente (Fig. 2.34). În funcție de necesitățile aplicației, ar putea fi definite diferite *lanțuri de comunicare*, în care unele faze de comunicare ar putea fi omise. De exemplu, nu este nevoie de condiționare atunci când se utilizează un senzor digital și nu este nevoie de detectarea și decodarea cadrelor pentru un senzor analog, dar toate componentele vor fi necesare pentru comunicarea digitală printr-un canal optic.

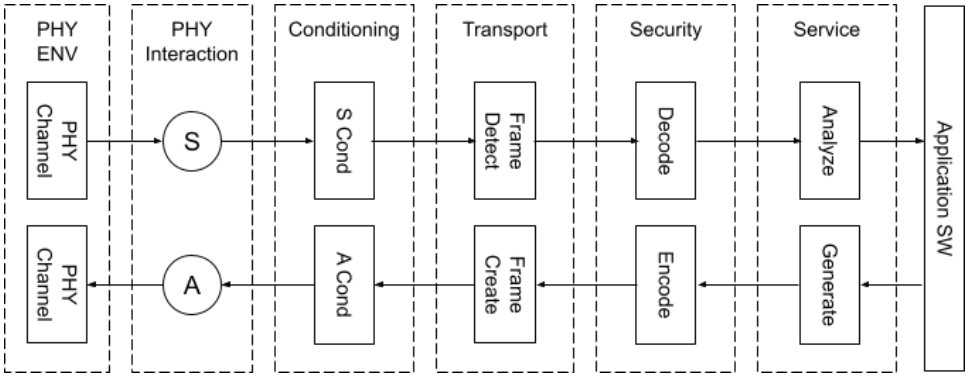


Fig. 2.34. Setul de componente pentru un lanț complet de comunicare [156]

În setul de componente ale *lanțului de comunicare*, semnalul trece prin multe faze similare, dar complementare pentru procesele de recepție și transmitere:

- Faza PHY-ENV reprezintă canalul de comunicație fizică, care interconectează sursa și destinația. Proprietățile canalului implică complexitatea următoarelor componente din lanțul de comunicare, cum ar fi condiționarea sau securitatea, dar nu numai.
- Faza de interacțiune PHY reprezintă conversia valorilor parametrilor fizici în informațiile interne ale sistemului și invers. Se compune din componente ale senzorului și ale actuatorului.
- Faza de condiționare reprezintă adaptarea informațiilor și filtrarea zgomotului colectate pentru a forma canalul sau procesele interne. În cazuri precum comunicarea digitală prin cablu pe același nivel de tensiune, această fază ar putea fi omisă.
- Faza de transport constă în detectarea și crearea de cadre, în conformitate cu un anumit model sau protocol. Această fază este responsabilă de transferul sigur al informațiilor prin canal.
- Faza de securitate reprezintă securizarea informațiilor și protejarea acestora împotriva accesului neautorizat. Informațiile sunt codificate la trimitere și decodificate la fluxul de primire.
- Faza de serviciu analizează informațiile primite, le pregătește pentru aplicare și generează informații, în conformitate cu cererea înaintată.
- Aplicația reprezintă expeditorul și receptorul fluxului de informații.

2.5.3 Modelul canale de comunicare

Structural, sistemul reprezintă o colecție de componente interconectate, care cooperează pentru a produce rezultatul [154]. Conform conceptului IoT ca dispozitiv electronic extins menționat mai sus, din punct de vedere fizic, componentele aceluiași sistem ar putea fi amplasate pe oricare dintre dispozitivele IoT, care sunt membre ale sistemului. Un sistem IoT este considerat un dispozitiv complet electronic, care imaginar a fost tăiat în părți, iar liniile de conectivitate rupte sunt înlocuite cu canale de comunicații. În acest fel, vor exista două tipuri primare de interconectări între componente, așa cum este prezentat în Fig. 2.35.

Canalul AB din Fig. 2.35 este un canal intern care conectează *Comp A* și *Comp B* de la aplicația care rulează pe *Dispozitiv AB*. Canalele interne sunt implementate ca interfețe software și comunicare între procese (IPC) [176].

Canalul AC conectează componentele *Comp A* și *Comp C* care fac parte din aplicații care rulează pe diferite dispozitive, *Dispozitiv A* și *Dispozitiv C*. Astfel de conexiuni ar putea fi implementate în urma setului de componente ale *lanțului de comunicare* prezentat în teză. Interfața componentei va arăta similar cu accesul canalului intern la o componentă, ca și cum s-ar afla pe același dispozitiv, dat fiind faptul existenței *lanțului de comunicare – Channel AC*.

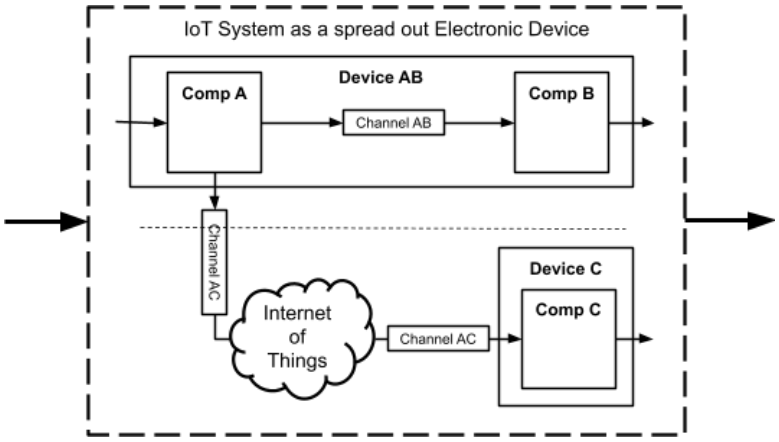


Fig. 2.35. Tipuri de canale de comunicare în sistem tip dispozitiv electronic distribuit [156]

Presupunând că canalele de comunicații abstractizează localizarea fizică a componentei, la nivel aplatizat, întregul sistem ar putea fi prezentat ca componente cu funcții de transfer interne interconectate cu canale, așa cum este ilustrat în Fig. 2.36.

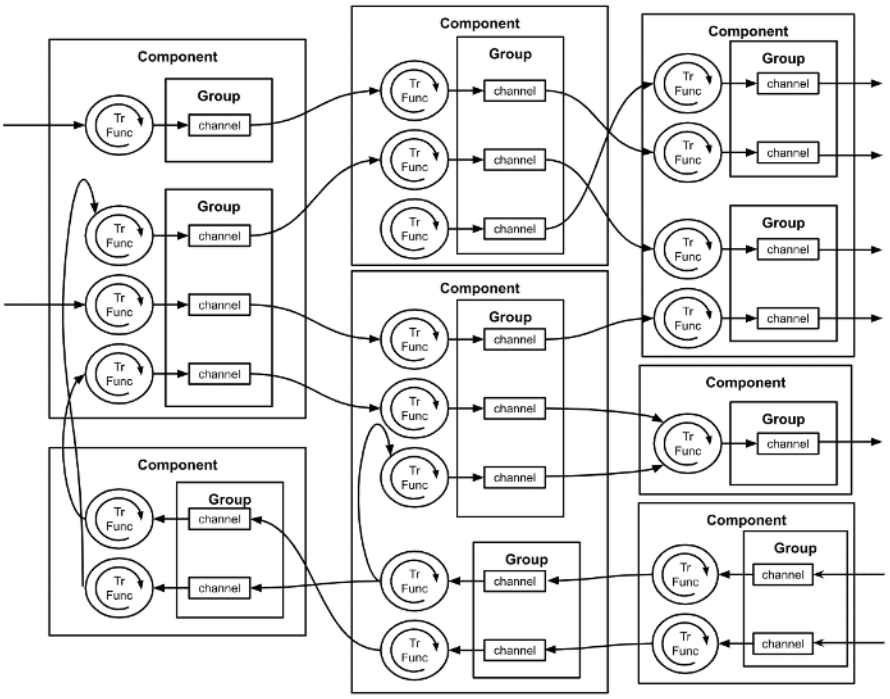


Fig. 2.36. Componente interconectate prin canale – vedere aplatizată [170]

O colecție de mai multe canale de semnal similare, care oferă valori pentru același domeniu sau parametru în mod logic, ar putea fi combinate într-un grup și operate împreună. În urma conceptului de abonat-editor, publicarea va actualiza semnalul componentei printr-o funcție de transfer, iar abonarea înseamnă conectarea la o interfață componentă specifică printr-un canal de comunicație. Termenul de a crea o legătură între componente este utilizat pentru procesul de abonare. În funcție de care capăt al canalului de comunicare joacă rolul de interlocutor activ, sunt posibile două tipuri de legături de canal – push sau pull, Fig. 2.37.

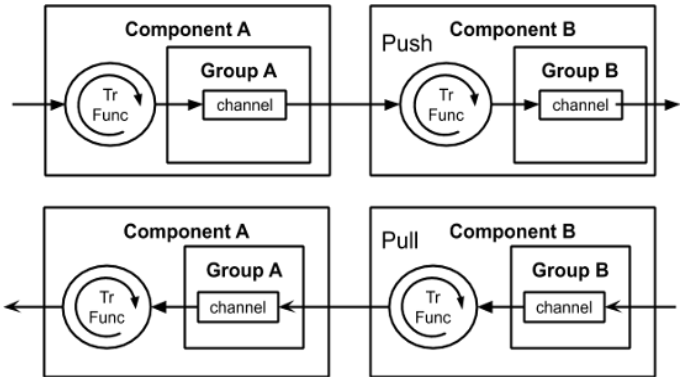


Fig. 2.37. Canale interconectate prin metode de tip push și pull [170]

O astfel de legătură implică o metodă *push* sau *pull* pentru a opera canalul cu scopul de a transmite sau a extrage informații. Aceeași metodă este utilizată pentru toate canalele incluse în același grup. Push înseamnă că canalele acceptă schimbarea valorii lor printr-o metodă specifică; Pull înseamnă că este evaluat intern și accesibil printr-o metodă la cerere. În [170] se propune o

metodă pentru dezvoltarea sistemelor încorporate prin configurații și generarea de cod bazată pe metamodele în formatul JavaScript – JSON, folosind conceptul de canal definit mai sus, precum și o propunere de instrument pentru operarea metodologiei.

2.6 Modelare sisteme prin meta descriere

2.6.1 Considerente arhitecturale

Conform teoriei sistemelor, un dispozitiv este considerat o colecție de componente care operează împreună pentru a produce un rezultat. Colectează informații din interfețele de intrare și furnizează rezultatul către interfața de ieșire. Fiecare componentă ar putea fi abstractizată ca model definit de o funcție de transfer care evaluează semnalele de intrare și furnizează semnalele rezultate (vezi Fig. 2.2). Interfețele de intrare, precum și cele de ieșire sunt deținătorii valorilor semnalelor care pot fi accesate în orice moment prin intermediul metodelor de citire a interfețelor.

În specificația AUTOSAR a driverului de intrări-ieșiri digitale (Digital Input Output – DIO), un pin IO digital de uz general reprezintă un canal DIO, iar un grup de canale este o combinație logică formală a mai multor canale DIO adiacente dintr-un port DIO [179]. Urmând metodele practicate din specificația AUTOSAR DIO, vom defini următoarele noțiuni:

- Channel – canal care reprezintă un semnal în componentă, accesibil printr-o interfață.
- Group – grup care este o colecție a mai multor canale de semnal similare din interfață, care furnizează valori pentru același domeniu sau parametru și reprezintă o grupare logică a canalelor în cadrul unei componente. În specificațiile AUTOSAR acest grup este limitat la un singur port fizic al microcontrolerului.
- Push / Pull – metode folosite pentru a opera canalul în scopul de a transmite sau extrage informațiile din canal. Aceeași metodă unică este aplicată tuturor canalelor incluse în grup. Metoda ar putea fi unul dintre tipurile Push sau Pull, unde Push înseamnă că respectivele canale acceptă schimbarea valorii printr-o metodă specifică externă, receptorul având un rol pasiv, iar Pull înseamnă că informația este extrasă, evaluată și accesibilă printr-o metodă internă la cerere, receptorul având un rol activ, Fig. 2.37.
- Componentă – o colecție de funcționalități sau metode reprezentate de funcții de transfer oferite de componentă și o colecție de canale care reprezintă tipurile de interfețe disponibile.
- Runnable – sarcină internă care definește comportamentul componentelor. Poate utiliza accesul canalului intern sau extern prin metode de tip push / pull atașate fiecărui grup de canale.

La un nivel aplatizat al dispozitivului, sistemul va arăta ca o colecție de canale care interacționează prin funcții de transfer, fiind divizate în grupuri și componente, Fig. 2.36.

2.6.2 Descrierea arhitecturii prin metamodele bazate pe JSON

Arhitectura sistemului poate fi descrisă prin definirea unui metamodel utilizând formatul JavaScript Object Notation – JSON [133]. Abordarea JSON permite definirea metamodelelor într-un mod simplu și clar. Acesta poate fi citit atât de utilizatorul uman, cât și suportat nativ de multe limbaje de programare, care ar putea fi folosite pentru a-l analiza, așa ca Python sau JavaScript. Intenția prezentată este de a avea o metodă foarte simplă, clara și ușor de întreținut pentru definiția arhitecturii, care poate fi implementată cu eforturi minime. Conform metodei prezentate în [170], în metodologie se propun trei tipuri de fișiere format JSON ale platformei: *manifest de platformă*, *definiții de componente* și *configurații de proiect*.

Manifest de platformă

Punctul de intrare al metodei de descriere este un fișier manifest în format JSON, în care sunt enumerate toate componentele disponibile în platformă pentru configurarea proiectului. Fișierul manifest „*platform.JSON”* este stocat într-un repozitoriu online, format Git și conține descrieri în formatul prezentat în exemplul de mai jos. Acest fișier reprezintă modalitatea de informare a utilizatorului platformei despre disponibilitatea resurselor în cadrul acesteia. Un exemplu al unui asemenea fișier este prezentat în Fig. 2.38, unde:

- "Description" – reprezintă descrierea generală a conținutului fișierului;
- "Type" – indicatorul tip al fișierului indică, în cazul dat, că fișierul conține referințe referitoare la componentele platformei.
- "Components" – este containerul tuturor referințelor componentelor disponibile în platformă;
- "<component_name>" – numele componentei urmată de descrierea acesteia;
- "git" – locația repozitoriului git pentru resursele componentei care conține codul sursă și alte resurse legate de componentă;
- "Path" – calea recomandată de amplasare a componentei în directoriul de proiect, care poate fi înlocuită în configurația JSON cu cea reală din proiect;

```
{
  "Description": "Platform manifest",
  "Type": "Platform",
  "git": "https://github.com/<file_path>,git",
  "Path": "TOOLS/app_builder/",
  "Components": {
    "<component_name>":{
      "git": "https://github.com/<file_path>.git",
      "Path": "MCAL/mcal_adc/"
    },
    "sensor": {
      "git": "https://github.com/<file_path>.git",
      "Path": "ESW/sensor/"
    }
  }
}
```

Fig. 2.38. Exemplu de manifest de platformă

Definiția de componentă

Fiecare componentă dezvoltată pentru utilizarea în platformă este asociată cu un fișier „*definition.JSON”* amplasat în directoriul principal al componentei. Definiția componentei conține constrângeri, recomandări, anumite asumări pentru configurarea componentelor proiectului. Definiția de componentă este un concept hibrid dintre manifest și definiție de tip de date. Conținutul său poate fi definit după exemplul din Fig. 2.39, unde:

- "Components" – conține definițiile tuturor componentelor incluse în fișierul de definiție;
- "<component_name>" – reprezintă numele componentei urmat de definiția acesteia;
- "git" – locația repozitoriului git pentru resursele componentei, care conține codul sursă și alte resurse legate de componentă;
- "Path" – calea recomandată în directorul de proiect, care poate fi înlocuită ulterior în configurația JSON a configurației componentelor;
- "Groups" – conține definițiile tuturor grupurilor din componentă;
- "<group_name>" – reprezintă numele grupului ce aparține unei componente urmat de definiția acestuia. De asemenea, este și definiția tip a grupului ca și referință în configurații;
- "NameSpace" – definește "<namespace_base>", baza pentru generarea incrementală automată a numelor de grupuri și canale în cadrul componentei care urmează aceeași definiție. În cazul în care nu este definit, numele de grup sau canal servește drept configurație substituentă a acestei setări;

```
{
  "Type": "Definition",
  "Components":{
    "<component_name>":{
      "git":"https://<path_to_git_file>.git",
      "Path": "<path_in_project>",
      "Groups":{
        "<group_name>": {
          "NameSpace": "<namespace_base>",
          "Multiplicity": "0-*",
          "Push": [<list_of_push_methods>],
          "Pull": [<list_of_pull_methods>],
          "Dependency": [
               <list_of_component_dependencies>
               ],
          "Channels": {
            "Multiplicity": "1-*",
               "NameSpace": "<namespace_base>",
            "Names": [<mandatory_names>]
               }
                },
          "Defines": {
            <component_parameter>:
                 [<purposed_value_list>]
          }
        }
    }
}
```

Fig. 2.39. Exemplu de definiție de componentă

- "Multiplicity" – reprezintă constrângerea de multiplicitate pentru grupuri sau canale care urmează aceeași definiție. Valorile de minimum și maximum sunt separate prin cratimă ”-” , iar „*” reprezintă infinitate. În cazul în care configurația este definită printr-un singur număr, aceasta indică că vor exista un număr strict de instanțe de grup sau canal. Când configurația nu este definită, valoarea acesteia va fi stabilită la configurația „0-*” – pot exista de la zero până la infinit instanțe de grupul sau canalul respectiv;
- "Push" – reprezintă o listă de metode de tip Push disponibile în cadrul componentei care pot fi folosite în configurații pentru accesul la canalele din grupul definiției date;
- "Pull" – reprezintă o listă de metode de tip Pull disponibile în cadrul componentei, care pot fi folosite în configurații pentru accesul la canalele din grupul definiției date;
- "Dependency" – listă de dependențe recomandate pentru stabilirea legăturilor dintre componente. Nu este obligatorie, și are ca scop sugerarea de conexiuni. Totodată, facilitează automatizarea descrierii configurațiilor;
- "Names" – listă de nume predefinite care vor fi selectate înainte de a genera nume de canale sau grupuri prin incrementare urmând configurația "NameSpace"
- "Defines" – definiții de parametri specifici componentei la care definiția respectivă este atașată, urmată de sugestii de valori specifice parametrului.

Definițiile sunt stocate împreună cu componentele, în directoriul de bază al acestora. În cazul în care definiția există, aceasta este utilizată în configurarea componentei în cadrul platformei, dacă nu, este generat un fișier nou de definiție după reguli generale, care poate fi adaptat în procesul configurării proiectului și adăugat la componenta de referință pentru facilitarea configurațiilor pentru proiectele viitoare.

Configurație de proiect

Configurația de proiect se descrie într-un fișier *<project_name>.JSON* localizat în directoriul principal al proiectului. Pentru demonstrarea metodei urmează un exemplu de interconectare a două componente, precum e arătat în Fig. 2.40.

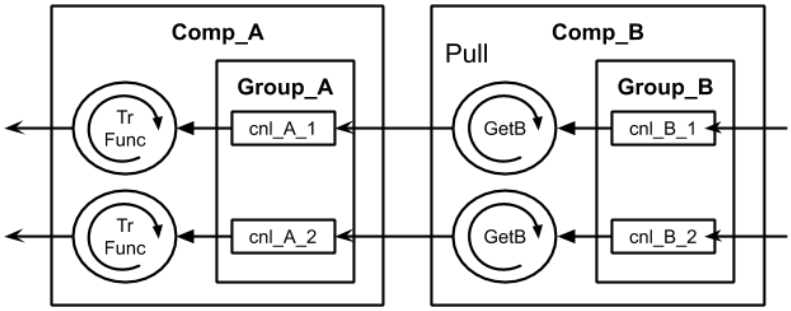


Fig. 2.40. Exemplu de interconectare a componentelor [170]

Conținutul fișierului de configurare va arăta ca în exemplul prezentat în Fig. 2.41.

```
{
 "Type": "Configuration",
 "Components":{
   "Comp_A": {                                    "Comp_B": {
     "git":"https://<path_to_git>.git",            "git":"https://<path_to_git>.git",
   "Path": "ESW/Comp_A/",                           "Path": "ESW/Comp_B/",
             "Groups": {                            "Groups": {
   "Group_A": {                                      "Group_B": {
     "Dependency":"Comp_B"                            "Dependency":"<a_dep_comp>"
     "Channels": {                                    "Channels": {
      "cnl_A_1": "cnl_B_1",                            "cnl_B_1": "<channel_link>",
      "cnl_A_2": "cnl_B_2",                            "cnl_B_2": "<channel_link>"
      },                                               },
     "Pull": "GetB"                                   "Pull": "<pull_method>"
     }                                                }
    }                                                }
   },                                               }
                                                   }
                                                  }
```

Fig. 2.41. Exemplu de definire de interconectare

unde:

- "Components" – conține descrierea configurațiilor tuturor componentelor din proiect;
- "Comp_A", "Comp_B": – numele componentelor din proiect urmate de descrierea configurațiilor acestora;
- "git" – locația repozitoriului git pentru resursele componentei, care conține codul sursă și alte resurse legate de componentă.
- "Path" – calea de amplasare a componentei în directorul de proiect;
- "Groups" - conține configurațiile tuturor grupurilor din componentă;
- "Group_A", "Group_B" – numele de grup urmate de configurațiile acestora;
- "Dependency" : "Comp_B" – indică componenta de care depinde grupul, respectiv, canalele din grup vor fi conectate către canale din componenta Comp_B definită ca dependență.
- "Channels" – conține configurațiile tuturor canalelor din cadrul grupului;
- "cnl_A_1": "cnl_B_1" – reprezintă configurația care indică că canalul cnl_A_1 din grupul Group_A al componentei Comp_A este conectat către canalul cnl_B_1 din grupul Group_B al componentei Comp_B;
- "Pull": "GetB" – în contextul exemplului de mai sus, configurația indică că canalul cnl_A_1 accesează date de la canalul cnl_B_1 prin metoda de tip pull GetB definită în componenta Comp_B;

În exemplul prezentat este utilizat un set minim necesar pentru a descrie o configurație, dar în scopuri de proiect ar putea exista și alte configurații, cum ar fi descrierea componentelor, tipurile componentelor, grupurilor sau altele, după necesitate.

Configurarea componentei din platformă

Metoda expusă în teză presupune că resursele componentei sunt compuse din trei părți: funcționalitățile componentei; adaptor componentă către platformă; configurația de interconectare a componentelor, prezentate în Fig. 2.42.

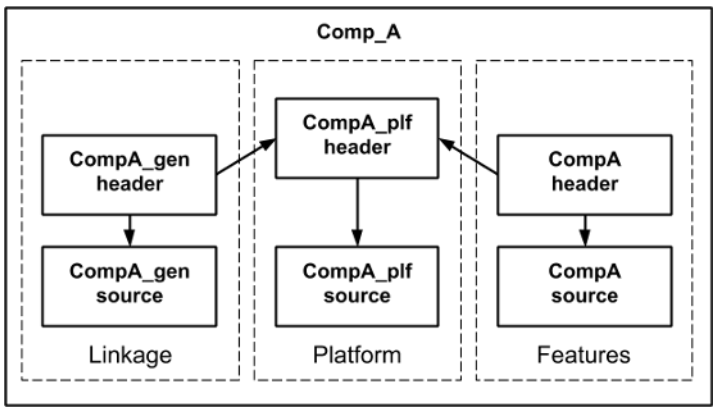


Fig. 2.42. Infrastructura internă a resurselor soft ale componentei [170]

Conform metodei, funcționalitățile componentei pot fi definite în cadrul procesului de proiectare sau preluate ca resurse externe, celelalte două fiind posibil de generat în procesul de dezvoltare a proiectului, utilizând conceptele platformei.

Funcționalitățile componentei – reprezintă o colecție de funcționalități, cum ar fi interfețe, funcții de transfer, modele comportamentale și altele specifice componentei date organizate în librării. Aceste funcționalități pot fi definite în cadrul procesului de dezvoltare a componentei sau preluate ca resurse terțe, ca cod sursă sau librării precompilate. Funcționalitățile din cadrul componentei sunt reprezentate de către un fișier antet C++, unde se află declarațiile acestora pentru a fi invocate. Prototipurile acestor funcționalități ar putea arăta ca în Fig. 2.43:

```
int ReadSpecificValue(); // în calitate de interfață
int EvaluateFunction(int in);// funcț. de transfer
int DoSpecificAction(); // model comportamental
```

Fig. 2.43. Exemplu de prototipuri de funcții prototip

Adaptare către platformă – este partea componentei care adaptează funcționalitățile disponibile ca o librărie specifică, pentru utilizarea acestora în cadrul platformei. Aici sunt localizate structurile de date de uz general specifice platformei, cum ar fi definiții de canale și grupuri, metode prin intermediul cărora acestea sunt gestionate urmând metodologia specifică platformei. Tot aici sunt definite valorile implicite pentru parametrii componentei, folosite în cazul în care nu sunt redefinite de configurația componentei.

Această abordare, separarea funcționalităților de partea de adaptare către platformă, permite ca funcționalitățile componentei să fie disponibile pentru utilizare în afara platformei, precum și resursele terților care să fie adaptate pentru a fi utilizate în platformă. În cazul

componentelor platformei, partea de adaptare vine la pachet cu componenta, iar în cazul adaptării resurselor terțe va fi necesar un efort suplimentar de definire a părții componentei pentru adaptare a resurselor la platformă.

Urmând exemplul a două interconexiuni de componente prezentate în secțiunea de definire a configurației din teză, setul minim de configurații specifice platformei pentru componenta *CompA* va arăta ca în Fig. 2.44 pentru antet și implementarea ca în Fig. 2.45

Toate datele legate de canal sunt stocate în descriptori de canale definiți printr-o structură de tip *COMP_A_CnlType*, unde sunt definite *linkCnlId* - ID-urile canalelor din *CompB,* urmând exemplul de mai sus utilizat pentru demonstrarea metodei; *ExtGetValue* – referință la metoda de tip pull definită în *CompB* folosită pentru citirea canalului său; *inputValue* – ultima valoare colectată prin conexiunea canalului; *outputValue* – ultima valoare evaluată a canalului.

Definiția *enum COMP_A_Cnl_IdType* este utilizată pentru definirea ID-urilor de canal dintr-un grup, ultimul identificator fiind numărul de canale din grup, similar pentru grupuri în componentă prin definiția de *COMP_A_Grp_IdType*.

```
#ifndef _COMP_A_PLF_H_
#define _COMP_A_PLF_H_

#include "comp_a_cfg.h"

#ifndef COMP_A_CONFIG
enum COMP_A_Cnl_IdType {COMP_A_CHANNEL_NR_OF = 0};
enum COMP_A_Grp_IdType {COMP_A_GROUP_NR_OF = 0};
#endif

typedef struct COMP_A_CnlType_t {
Std_CnlIdType linkCnlId = 0;
Std_PullType ExtGetValue = NULL;
Std_PhyDataType inputValue;
Std_PhyDataType outputValue;
} COMP_A_CnlType;

COMP_A_CnlType* COMP_A_GetCnlRef (…);
Std_ReturnType COMP_A_CnlSetup(…);
Std_ReturnType COMP_A_GroupSetup(…);
Std_ReturnType COMP_A_SetPullMethod(…);
Std_ReturnType COMP_A_SetGroupPullMethod(…);

Std_PhyDataType COMP_A_GetValue(Std_CnlIdType);

#endif
```

Fig. 2.44. Antetul platformei *comp_a_plf.h*

```
#include "comp_a_plf.h"

COMP_A_CnlType COMP_A_Cnls [COMP_A_CHANNEL_NR_OF];
COMP_A_CnlType* COMP_A_Grps [COMP_A_GROUP_NR_OF];

COMP_A_CnlType* COMP_A_GetCnlRef (…){…}
Std_ReturnType COMP_A_CnlSetup(…){…}
Std_ReturnType COMP_A_GroupSetup(…){…}
Std_ReturnType COMP_A_SetPullMethod(…){…}
Std_ReturnType COMP_A_SetGroupPullMethod(…){…}

Std_PhyDataType COMP_A_GetValue(Std_CnlIdType Id){
/* exemplu a unei funcţii de tip pull */
 COMP_A_CnlType* cnlRef = COMP_A_GetCnlRef(Id)
Std_PhyDataType input, result;
input = cnlRef->ExtGetValue(cnlRef->linkCnlId)
result = EvaluateFunction(input);
return result;
}
```

Fig. 2.45. Implementare *comp_a_plf.c*

Instanțele canalelor din componentă sunt definite de *COMP_A_Cnls [COMP_A_CHANNEL_NR_OF],* care reprezintă o listă de descriptori, structuri, de canale pentru un grup, accesibil prin ID-ul său, iar instanțele grupurilor sunt definite de *COMP_A_Grps [COMP_A_GROUP_NR_OF]* – reprezentând o listă de grupuri din componenta care conține referințe la instanțele grupului de canale. În mod implicit, dacă nu este aplicată nicio configurație, listele de canale vor fi de mărime zero, dar după includerea configurațiilor, acestea vor fi redefinite.

În fiecare componentă sunt definite un set de metode-funcții pentru gestionarea canalelor, cum ar fi – *COMP_A_GetCnlRef* pentru extragerea referinței la canal prin ID-ul său; *COMP_A_CnlSetup* pentru interconectarea canalelor, în exemplul nostru pentru înregistrarea ID-urilor canalului **CompB** în descriptorii canalului *CompA*; *COMP_A_GroupSetup* pentru stabilirea conexiunilor între canalele a două grupuri; *COMP_A_SetPullMethod* pentru înregistrarea metodei de componentă tip pull în descriptorul canalului; *COMP_A_SetGroupPullMethod* pentru înregistrarea metodei de tip pull pentru întregul grup.

Implementarea metodei *COMP_A_GetValue* va folosi metodele din setul de funcționalități ale componentelor, dar și metode de interfață disponibile prin referințe de tip pull, cum ar fi *ExtGetValue*, din descriptorul canalului.

Configurația componentei presupune cererea de fișiere de configurații, antet și implementare, urmând descrierile din fișierele de configurație a proiectului *<project_name>.JSON* unde se regăsesc configurațiile pentru toate componentele proiectului.

Urmând exemplul de referință, pentru interconectarea a două componente, în calitate de configurație a componentei *CompA* pentru fișierul antet *comp_a_cfg.h* vom avea un cod sursă generat, după cum e prezentat în Fig. 2.46.

```
#ifndef _PROJECT_CONFIG_H_
#define _PROJECT_CONFIG_H_
#include "platform_config.h"

#define COMP_A_CONFIG
enum COMP_A_Cnl_IdType {cnl_A_1, cnl_A_2, COMP_A_CHANNEL_NR_OF};
enum COMP_A_Grp_IdType {Group_A, COMP_A_GROUP_NR_OF};
#include "ESW/Comp_A/Comp_A.h"

#define COMP_B_CONFIG
enum COMP_B_Cnl_IdType {cnl_B_1, cnl_B_2, COMP_B_CHANNEL_NR_OF};
enum COMP_B_Grp_IdType {Group_B, COMP_B_GROUP_NR_OF};
#include "ESW/Comp_B/Comp_B.h"

Std_ReturnType Project_config(void);
```

Fig. 2.46. Configurație a componentei *CompA* pentru fișierul antet *comp_a_cfg.h*

Acest fișier antet de configurații fiind inclus în fișierul antet al componentei de platformă va redefini configurațiile implicite pentru grupuri și canale, inițial fiind definite de mărime zero.

Pentru fișierul implementare *comp_a_cfg.c* vom avea funcția de conectare a componentelor prin stabilirea de conexiuni între canale, după cum urmează în Fig. 2.47.

Prin invocarea funcției *CompA_config* în cadrul proiectului se for stabili conexiunile între componentele **CompA** și **CompB** prin intermediul canalelor **cnl_A_1** și **cnl_A_2** conectate respectiv cu canalele **cnl_B_1** și **cnl_B_2** transferul de informație fiind realizat prin metoda de tip pull **GetB**, pe care **CompA** o accesează prin referință din descriptorul canalului **ExtGetValue** definit ca structură de tip **COMP_A_CnlType**.

```
#include "comp_a_cfg.h"
Std_ReturnType CompA_config(void)
{
Serial.begin(115200);
Serial.println("ES Platform based Project");
Std_ReturnType error = E_OK;

error += COMP_A_ChannelSetup(cnl_A_1, cnl_B_1);
error += COMP_A_ChannelSetup(cnl_A_2, cnl_B_2);

error += COMP_A_SetPullMethod(cnl_A_1, GetB);
error += COMP_A_SetPullMethod(cnl_A_2, GetB);
Serial.print("GROUP_A configured - Error : ");
Serial.println(error);

return error;}
```

Fig. 2.47. Configurație conexiuni între canale

Automatizarea procesului de configurare a proiectelor presupune utilizarea de programe specializate, care permit gestionarea fișierelor de manifest platformă, de definiții ale componentelor și configurațiilor acestora, excluzând intervenții la nivel text pentru a exclude erorile de editare. Utilizarea acestor resurse de programe permite ca prin intermediul interfețelor grafice de a genera configurații utilizând metode de selecție din resursele recomandate în baza informației colectate din multitudinea de fișiere tip JSON urmând metodologia din teză.

2.7 Concluzii la capitolul 2

În urma analizei aspectelor cu privire la *metode de modelare a sistemelor electronice distribuite* s-au stabilit următoarele:

1. Din punctul de vedere al sistemelor, sistemele electronice distribuite interacționează cu mediul înconjurător, utilizatorul și alte dispozitive.
2. A fost introdusă noțiunea de *arhitectură generică* pentru un sistem electronic distribuit, unde funcționalitățile se pot grupa în componente generice pentru achiziții de semnale, acționare asupra mediului, comunicare, interacțiune cu utilizatorul, stocare de date, gestionare energie și componente specifice aplicației.
3. S-a constatat că componentele generice ale dispozitivului electronic pot fi modelate cu *arhitecturi în straturi* similare, acoperind aspectele atât hardware, cât și software.
4. S-a demonstrat că funcționalitățile componentelor de achiziție și a celor de acționare asupra mediului sunt complementare și pot fi modelate prin *fluxuri de date* cu *funcții de transfer* similare pentru condiționarea semnalelor.
5. S-a realizat o asociere a fluxului de diagnosticare și monitorizare al sistemului cu fluxul de achiziție semnal din mediul fizic, iar cel de protecție cu fluxul de acționare asupra mediului, funcționalitățile de diagnoză și protecție considerându-se la fel complementare.
6. A fost arătat că prin interacțiunea între componentele de diagnoză și protecție se poate asigura mecanisme de siguranță a funcționarii dispozitivului, precum și a siguranței mediului înconjurător față de acțiunile sau comportamentul sistemului.
7. S-a constatat că comunicarea între dispozitivele electronice ale unui sistem electronic distribuit se supune acelorași reguli ca și în comunicarea interpersonală.
8. S-a arătat cum funcționalitățile ce compun fluxurile de informație în componentele pentru achiziție, acționare și comunicare pot fi organizate în comun pentru a forma un concept de *lanțuri de comunicare* complexe, funcționalitățile de bază a acestor componente considerându-se ca și caz particular al acestui concept.
9. A fost introdus conceptul de *lanț de comunicare* care permite modelarea de interacțiuni complexe cu funcționalități din setul de interacțiune cu mediul, condiționare, transport, securitate și diverse servicii.
10. S-a constatat că modelarea cu *lanțuri de comunicare* poate fi automatizată cu resurse specializate de produs program prin definirea de configurații de interacțiuni de funcții de transfer formate din funcționalitățile componentelor generice, dar și a celor de aplicație.

11. S-a arătat cum componentele din stratul ASW pot fi realizate cu modele matematice, funcționale de control sau modele comportamentale specifice aplicației, construite în baza modelării cu *lanțuri de comunicare* complexe.
12. A fost introdusă metodologia de definire a componentelor configurabile ce implică beneficii de flexibilitate în modelarea sistemelor, dar și de reutilizare și adaptare a resurselor de program deja existente, fapt care reduce timpul de obținere a soluțiilor pentru realizarea proiectelor.
13. S-a demonstrat că conceptul de dispozitiv electronic distribuit permite abstractizarea sistemelor de tip IoT ca fiind un singur dispozitiv cu un acces asigurat la orice de oriunde – Everything from Everywhere.

3. MODELAREA ARHITECTURALĂ A SISTEMELOR ELECTRONICE DISTRIBUITE. STUDII DE CAZ

3.1 Procesul de modelare a sistemelor electronice distribuite

Procesul de dezvoltare a sistemelor electronice distribuite, sau sisteme incorporate, este un proces complex și necesită abordări de modelare arhitecturală și specifice disciplinelor din ingineria mecanică, electrică/electronică și software. Acest proces poate fi împărțit în mai multe etape, fiecare cu propriile sale cerințe, considerente și instrumente de modelare.

Printre cele mai importante etape ale procesului de dezvoltare a sistemelor electronice distribuite se regăsesc cele prezentate în Fig. 3.1.

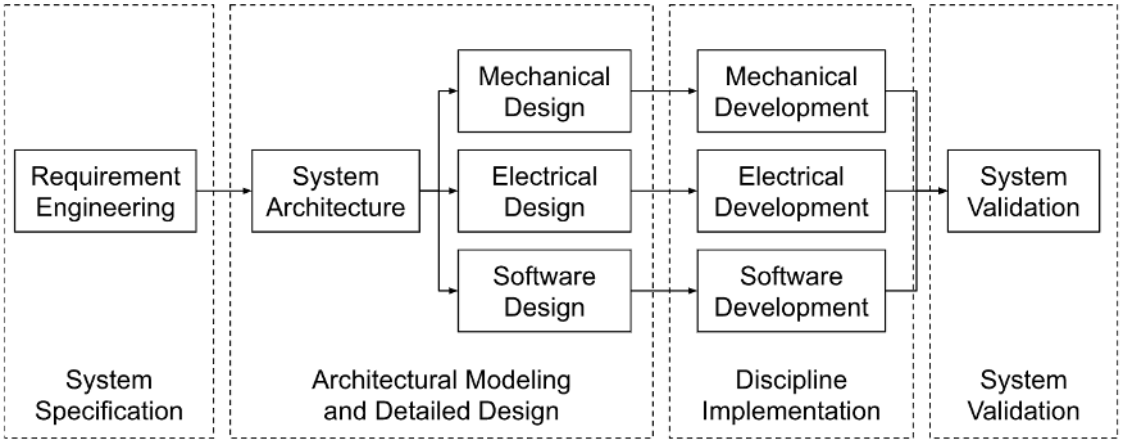


Fig. 3.1. Procesul de dezvoltare a sistemelor electronice distribuite

System Specification – reprezintă etapa inițială care implică colectarea și analiza cerințelor utilizatorilor, identificarea obiectivelor sistemului și stabilirea constrângerilor de proiectare. Unele dintre instrumentele utilizate la această etapă includ diagramele UML, diagramele de flux de date și brainstorming-ul. Ca rezultat al acestei etape este definirea specificațiilor sistemului.

Architectural Modeling reprezintă etapa de proiectare arhitecturală a sistemului sau inginerie de sistem și implică definirea arhitecturii sistemului, specificarea componentelor hardware și software, precum și stabilirea interacțiunilor dintre acestea.

Modelarea arhitecturală în straturi cu *lanțuri de comunicare* prin intermediul fișierelor în format JSON reprezintă un element cheie în metodologia descrisă în teză în capitolul 2. Metodologia propusă în această lucrare a fost elaborată în baza experienței acumulate în multiple elaborări de sisteme de diversă complexitate ale autorului. Conceptul de modelare arhitecturală de sistem prin definirea *lanțurilor de comunicare* a fost folosit în studiile de caz prezentate în continuare în teză, în mod special, pentru sistemul de monitorizare a mediului, sistem de control al brațelor robotice, sistem de control al reflecției luminii și sistem de control al camerei de uscare a fructelor. Rezultatele modelării sistemelor menționate sunt prezentate în secțiunea 3.3 al acestei

teze. Unele dintre colecțiile de configurații de *modele arhitecturale în straturi* cu *lanțuri de comunicare* pentru diverse proiecte menționate în teză sunt prezentate în Anexa 4.

Configurațiile interconectare a componentelor proiectului în formă de definiții, interconexiuni și funcții de interconectare în format C/C++ sunt generate în baza *modelului arhitectural* în format JSON. Un exemplu de configurație în format JSON pentru sistemul de control al brațelor robotice 3DOF este prezentat în Anexa 5.

Detailed Design reprezintă modelarea arhitecturală de detaliu în cadrul disciplinelor de inginerie mecanică, inginerie electrică și inginerie software implicate în proiectarea sistemului electronic distribuit.

În disciplina de inginerie software, proiectarea sistemelor electronice distribuite poate include proiectarea și dezvoltarea aplicațiilor software pentru controlul și gestionarea sistemului. Aceasta poate fi efectuată prin utilizarea limbajelor de programare, cum ar fi Java, Python sau C++, și a mediilor de dezvoltare integrat, cum ar fi Eclipse, Visual Studio sau NetBeans.

Aplicând metodologia expusă în teză, efortul principal îi revine dezvoltării aplicației conform specificației sistemului prin definirea funcționalităților componentelor funcționale sau comportamentale al sistemului. Procesul de proiectare a sistemului de control al unui frigider inteligent făcând abstracție de platforma care realizează interacțiunea cu mediul extern este prezentat în Anexa 1.

În multe cazuri funcționarea unui sistem se supune unui model matematic, care urmează a fi încorporat în sistemul electronic. În procesul de modelare matematică intervin constrângeri și factori specifici sistemului, care sunt luate în considerație în procesul modelării. Asemenea considerentele de modelare matematică pentru un sistem de poziționare a oglinzilor cu redirecționarea luminii reflectate de la soare sunt prezentate în Anexa 3.

Procesul de implementare a sistemului implică dezvoltarea software-ului și a componentelor hardware ale sistemului, precum și integrarea acestora într-un produs final funcțional. Exemple de instrumente utilizate în această etapă includ medii de dezvoltare integrat (IDE), precum Visual Studio sau Eclipse, și sisteme de control al versiunii, precum Git. Metodologiile utilizate pot fi Extreme Programming sau DevOps.

Pentru a ușura compatibilitatea între componente este necesară o compatibilitate la nivel de interacțiune între ele. Acest fapt se poate realiza prin a defini un template standard de componentă și utilizarea acesteia ca cod inițial în procesul de dezvoltare. În teză s-au introdus considerente de generare a componentelor de platformă în scop de a asigura această compatibilitate între componente, precum și ușurarea procesului de dezvoltare a acestora. Un exemplu de

componentă de platformă generată în baza de template comparată cu varianta finală de implementare a acesteia este prezentat în Anexa 6.

În disciplina de inginerie electrică/electronică, proiectarea sistemelor electronice distribuite poate include proiectarea și dezvoltarea circuitelor electronice, microcontrolere și senzori. Acestea pot fi realizate prin utilizarea software-ului de simulare și proiectare a circuitelor, cum ar fi Altium Designer, PSpice sau Proteus. De asemenea, proiectarea și dezvoltarea sistemelor de comunicații wireless, precum WiFi sau Bluetooth, pot fi incluse în această etapă.

Conform principiilor de modelare a sistemelor electronice distribuite, luând în considerație *arhitectura în straturi*, componentele hardware presupun o proiectare în domeniul ingineriei electrice și electronice. De regulă, componentele electronice funcționale sunt proiectate specific pentru fiecare dispozitiv electronic. Chiar dacă nu este posibilă o proiectare complet automatizată, în cazul componentelor realizate în domeniul ingineriei electrice se intervine cu recomandări de abordare reieșind din experiența acumulată în cadrul altor proiecte. Proiectarea schemelor electrice pentru proiectul uscătoriei de fructe care pot fi documentate și preluate ca și recomandări de proiectare pentru alte proiecte este prezentată în Anexa 2.

Componentele de asigurare cu energie electrică a sistemului din punct de vedere al funcționalității sistemului pot fi considerate auxiliare, deoarece au implicații neesențiale în funcționalitățile de bază ale sistemelor. Funcția acestor componente este una vitală, de asigurare cu energie electrică a sistemului și asigurarea funcționarii normale din punct de vedere al alimentării pentru componentele funcționale. La acest nivel sistemul de asigurare cu energie electrică este redus la proiectarea distribuției energiei de la rețea către componentele instalației, fără a fi elaborată o interacțiune esențială de control proiectată la nivel software. Proiectarea sistemului de distribuite a energiei electrice pentru instalația de uscare a fructelor este reprezentată în Anexa 9.

În disciplina de inginerie mecanică, proiectarea sistemelor electronice distribuite poate include simularea, proiectarea și analiza mecanică a carcasei și a componentelor, cum ar fi montarea plăcii de circuit.

Considerentele mecanice ale sistemelor stau la baza soluțiilor de funcționalități mecanice ale sistemului la nivel de interacțiune cu mediul exterior, ele fiind în afara scopului acestei teze. În procesul de proiectare aceste considerente își aduc impactul prin impunerea de constrângeri de proiectare în domeniul ingineriei electrice și software prin intermediul parametrilor mecanici ai construcției. Considerentele mecanice ale instalației de uscare a fructelor, care urmează a fi luate în considerație în procesul de la modelare electronică și electrică, dar și la modelarea componentelor software a sistemului, prezentate în Anexa 8.

Validarea sistemului – această etapă implică verificarea funcționalității, fiabilității și performanței sistemului. Exemple de instrumente utilizate în această etapă includ automatizarea testelor cu Selenium sau JUnit, precum și sisteme de gestionare a defectelor, precum JIRA sau Bugzilla. Metodologiile utilizate pot fi Test Driven Development (TDD) Behavior Driven Development (BDD). Validarea conceptelor propuse în teză se realizează prin intermediul testelor de integrare, adică validarea acestora prin identificarea funcționalității acesteia după integrarea în proiect. La această etapă validarea se face în mare parte prin revizuire și observație, însă pe viitor se planifică implementarea metodologiilor TDD și BDD.

Ca urmare, procesul de dezvoltare a sistemelor electronice distribuite poate fi abordat prin intermediul mai multor faze distincte, fiecare având propriile instrumente și metodologii specifice. Cu toate acestea, este important să se adapteze procesul la problema reală. Totodată, dezvoltarea sistemelor electronice distribuite implică o varietate de discipline din ingineria mecanică, electrică/electronică și software. Fiecare dintre aceste discipline aduce propriile sale considerente și instrumente în cadrul fiecărei faze a procesului de dezvoltare, care poate aduce beneficii semnificative în ceea ce privește scalabilitatea, flexibilitatea și securitatea sistemului, și poate fi realizat prin utilizarea de metode și tehnologii moderne, dezvoltarea căreia ~~și~~ este unul din obiectivele principale expuse în teză.

3.2 Modelarea sistemelor electronice distribuite în baza produsului program

Reutilizarea resurselor de program este esențială în ritmul actual de dezvoltare a tehnologiilor informaționale. Dezvoltarea de soluții bazate pe resurse reutilizabile, față de cele dezvoltate în totalitate manual în cadrul procesului de proiectare, aduc numeroase avantaje, printre care reducerea timpului de realizare, stabilitatea sistemului, modularitate și altele esențiale. Un avantaj îl aduc platformele de dezvoltare de proiecte prin configurare și generare cod, implicând reducerea efortului de adaptare a resurselor reutilizabile în cadrul proiectului și configurarea formală a acestora. În urma utilizării unor asemenea tehnologii de automatizare, efortul de dezvoltare revine satisfacerii cerințelor funcționale ale aplicației de nivel înalt, descriptiv, prin utilizare de metode formale sau metalimbaje.

În teză au fost elaborate: o metodă de localizare a resurselor reutilizabile prin intermediul fișierului de manifest al platformei; interconectare după un principiu comun întregii platforme; adaptarea resurselor terțe către mecanismele de interconectare adoptate în cadrul platformei. De asemenea, s-a realizat o bază de resurse de componente, sub forma unei platforme online, pentru automatizarea procesului de proiectare în baza unor recomandări din fișierele de definiție a

componentelor reutilizabile, fapt care a deschis noi orizonturi și provocări pentru dezvoltarea platformei prezentate în teză.

În scopul demonstrării viabilității metodei prezentate a fost elaborat un produs program specializat pentru crearea și configurarea proiectelor pentru sisteme încorporate.

Produsul software menționat este compus din trei secțiuni: secțiunea de gestionare a platformei, secțiunea de gestionare a configurațiilor componentelor modelului și secțiunea de modelare a proiectului.

Ca și rezultat al produsului program se vor evidenția *Generarea configurației conform modelului, Generarea arhitecturii conform modelului proiectului* și *Generarea resurselor de proiect conform modelului definit*.

Secțiunea de gestionare a platformei

Această secțiune a resursei de program este dedicată gestionării componentelor înregistrate în platforma online, referințele cărora sunt definite într-un fișier format JSON. Inițial, fișierul este definit pe stația locală, după care în conformitate cu configurațiile din acest fișier de configurație este transferat pe platformă online, și populat pe parcursul modelarii în cadrul mai multor proiecte. Fiecare proiect își aduce aportul său pentru dezvoltarea platformei prin îmbogățirea acesteia cu noi componente pentru reutilizare în alte proiecte, fapt care are un impact de reducere considerabilă a timpului de realizare a proiectelor.

Elementele principale ale fișierului de configurație a platformei sunt: adresa de referință a platformei online, o listă de componente definite prin numele lor, referințele repozitoriilor online ale componentelor, localizarea recomandată în cadrul proiectului. Un exemplu al acestei configurații se poate vedea în Fig. 3.2.

În cadrul resursei de program acest fișier poate fi vizualizat grafic, cum este prezentat în Fig. 3.3. În această secțiune se pot vizualiza toate componentele listate, disponibile pentru modelare, localizate după o adresă online. În cazul dat, particular, după adresa: https://github.com/ML-ES-Platform într-un fisier format JSON – ”components.json”.

În partea stângă a secțiunii vor fi listate componentele de pe platformă care se vor clasifica în patru categorii de referințe către componente, fiecare marcata după codul culorilor:

- ”green” – componente disponibile în platformă și utilizate în model colorate în verde.
- ”purple” – componente disponibile în platformă, dar neutilizate în proiect, colorate în roșu-violet.
- ”red” – componente utilizate în model, dar nedisponibile în platformă, colorate în roșu.

- ”violet” – componente referențiate ca și dependențe, dar care nu sunt disponibile în platformă și nici în model colorate în violet.

```
{
    "Description": "Embedded System Platform manifest",
    "Type": "Manifest",
    "git": "https://github.com/ML-ES-Platform/app_builder",
    "Path": "TOOLS/app_builder/",
    "Components": {
        "mcu": {
            "TypeName": "mcu"
        },
        "mcal_adc": {
            "TypeName": "mcal_adc",
            "git": "https://github.com/ML-ES-Platform/mcal_adc.git",
            "Path": "MCAL/mcal_adc/"
        },
        "dd_potentiometer": {
            "TypeName": "dd_potentiometer",
            "git": "https://github.com/ML-ES-Platform/dd_potentiometer",
            "Path": "ESW/dd_pot/"
        },
        "vd_angle_sens": {
            "TypeName": "vd_angle_sens",
            "git": "https://github.com/ML-ES-Platform/vd_angle_sens.git",
            "Path": "ESW/vd_angle_sens/"
        },
        "os_time_trig": {
            "TypeName": "os_time_trig",
            "git": "https://github.com/ML-ES-Platform/os_time_trig.git",
            "Path": "BSW/os_time_trig/"
        },
        "arm_6dof": {
            "TypeName": "arm_6dof",
            "git": "https://github.com/ML-ES-Platform/arm_6dof.git",
            "Path": "ASW/arm_6dof/"
        }
    }
}
```

Fig. 3.2. Exemplu de configurație de platformă

În cazul în care există o definiție a componentei, setările acesteia se cor vizualiza, în partea dreaptă a secțiunii, în caz contrar, când nu există, acestea vor trebui să fie definite local și, eventual, transferate pe platforma online pentru referențiere un următoarele proiecte.

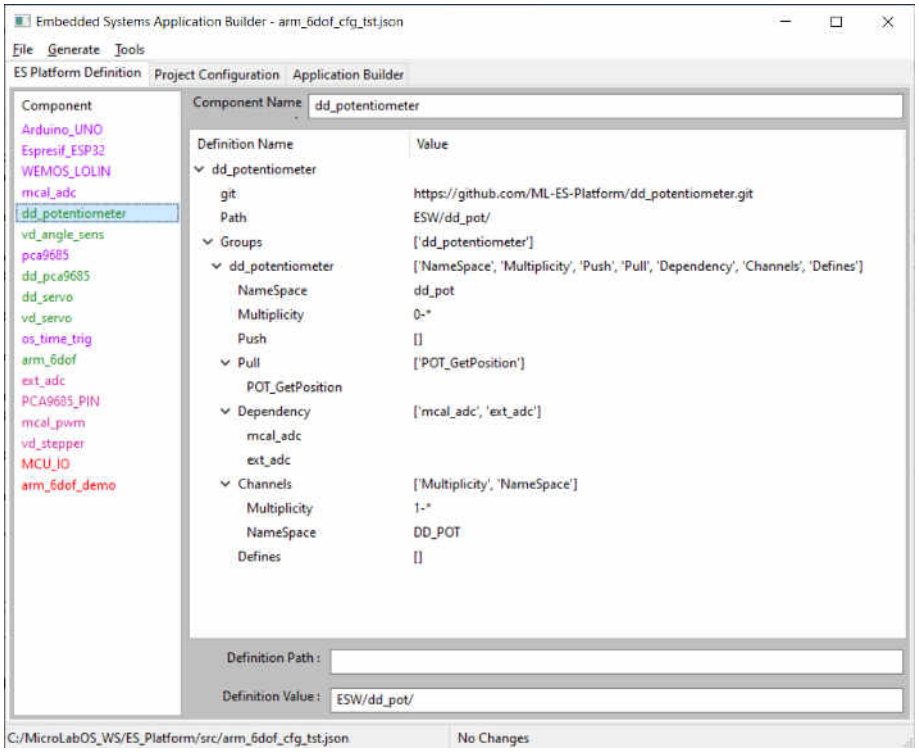


Fig. 3.3. Gestionarea definițiilor componentelor [170]

Secțiunea de gestionare a configurațiilor componentelor modelului

Această secțiune a resursei de program este dedicată gestionării configurațiilor componentelor din model definite într-un fișier format JSON. Fișierul de configurare este localizat în cadrul structurii de fișiere de proiect și conține: o listă de componente definite prin nume, referințele repozitoriilor online ale componentelor, o localizare recomandată în cadrul proiectului. Fiecare componentă conține configurațiile tuturor interacțiunilor între componente prin canale organizate în grupuri.

În resursa de program în coloana din partea stângă sunt localizate componentele din cadrul modelului proiectului, iar în partea dreaptă – configurația componentei selectate, Fig. 3.4.

În cazul disponibilității componentelor online acestea sunt importate, dacă nu, se creează o componentă dintr-un template care poate fi populat, și pe final adăugat la platforma online. La fel ca și în secțiunea componentelor de platformă, culorile vor indica disponibilitatea resurselor reutilizabile. Pe măsură ce metodologia expusă în teză va fi utilizată în mai multe proiecte, componentele neexistente în platforma vor fi definite inițial în cadrul proiectului, iar după înregistrare în cadrul platformei și vor trece în starea de ”green”, adică componențe disponibile în platformă și utilizate în model.

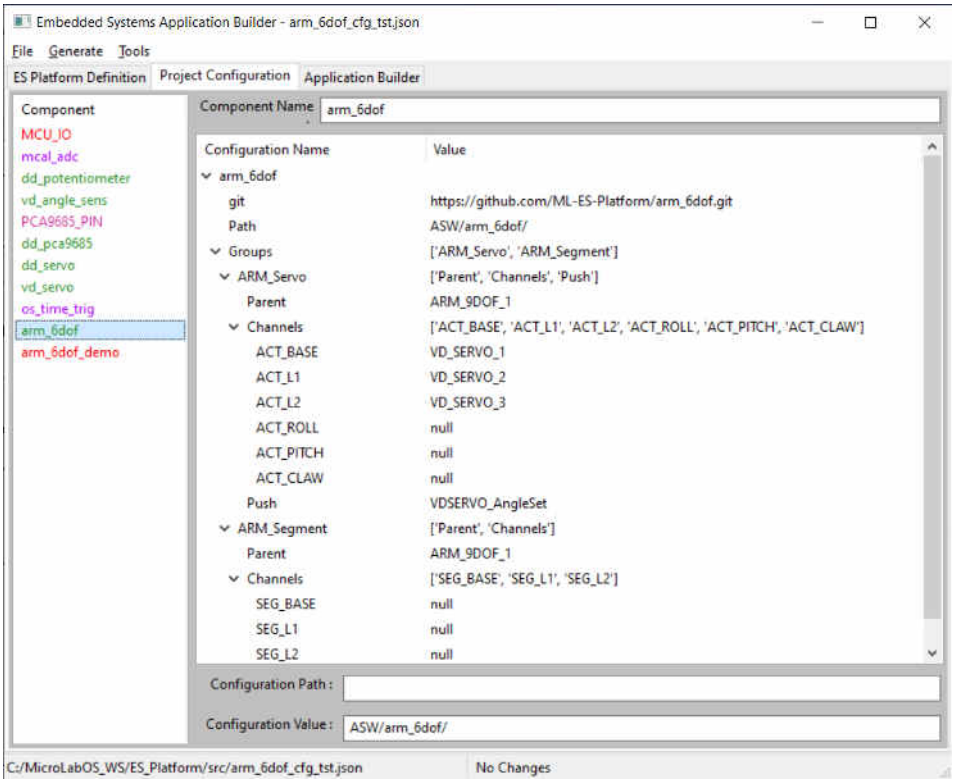


Fig. 3.4. Gestionarea configurațiilor componentelor [170]

Secțiunea de modelare a proiectului

Produsul de program elaborat permite vizualizarea și editarea definițiilor și a configurațiilor componentelor importate din cadrul unei platforme de resurse stocate online și accesibile pentru realizarea de proiecte pentru dispozitive de tip sistem încorporat. În scopul de generare automată a fișierelor sursă de configurare a componentei, produsul de program are facilități de interconectare vizuală a componentelor pentru automatizarea procesului de proiectare. Interfața grafică pentru secțiunea de configurare a proiectului este prezentată în Fig. 3.5.

Această secțiune permite editarea canalelor din componenta sursă selectată, coloana de componente din stânga, în raport cu componenta destinație, coloana de componente din partea dreaptă. Respectiv, prin intermediul acestei secțiuni a resursei de program se vor adăuga canale organizate în grupuri, cărora li se vor defini referințe către canalele componentei destinație, indicând și funcția de transfer utilizată pentru extragerea sau transmiterea informației.

Câmpurile de configurare pentru stabilirea legăturilor între componente prin canale propun recomandări colectate din configurațiile componentelor proiectului, dar și din dependențele implicite. Totodată, permit de a introduce definiții noi de canale, grupuri și componențe. De menționat că componentele noi introduse prin această metodă se vor clasifica ca componente din categoria ”red” - componente utilizate în model, dar nedisponibile în platformă, respectiv va impune o necesitate de a defini această componentă. Eventual, prin intermediul produsului de

program, această componentă se va completa și se va defini conform conceptului definit în teză. Experiența acumulată în diverse aplicații proiectate prin această metodă de modelare va conduce la rafinarea componentelor și completarea cu noi recomandări de definire de legături cu canale de transfer informație.

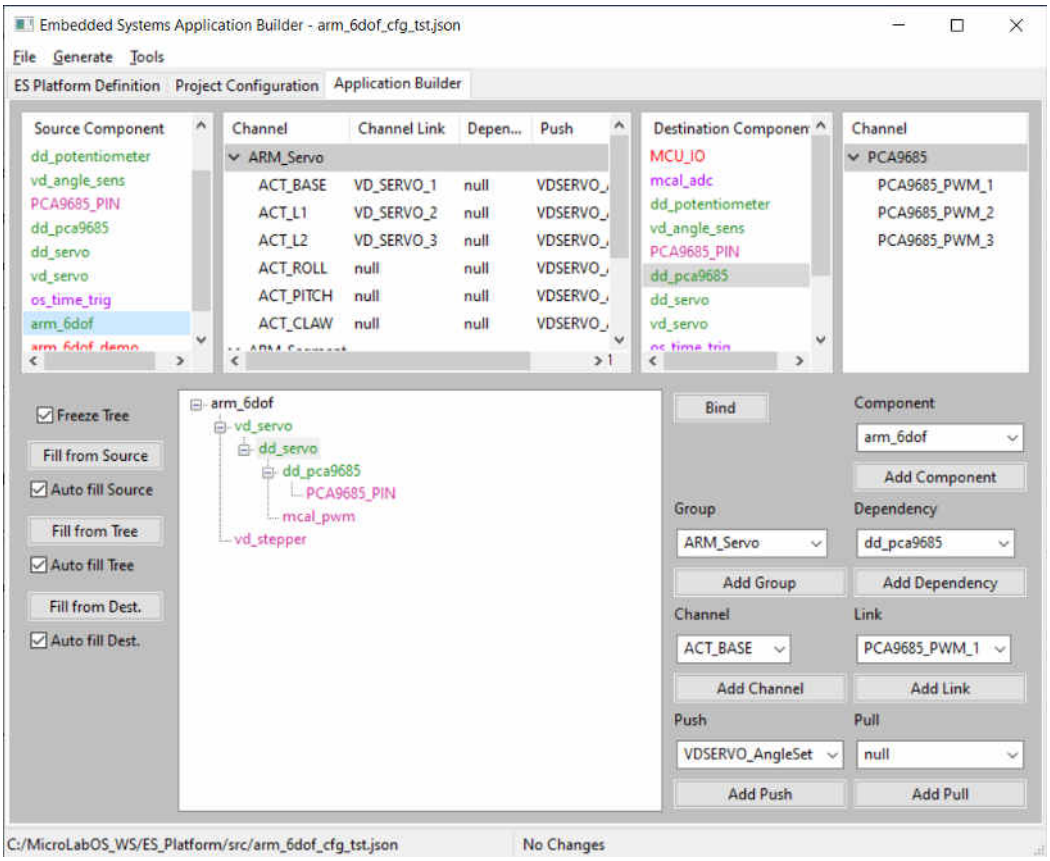


Fig. 3.5. Gestionarea interconexiunilor componentelor [170]

Există și posibilitatea de a explora proiectul prin ierarhia de componente, unde se pot vizualiza alternative de stabilire a legăturilor. De exemplu, în Fig. 3.5 componenta de dd_servo care folosește dd_pca9685 pentru generare semnal PWM, ca și alternativă pentru generare de semnale PWM poate servi componenta mcal_pwm.

Generarea configurației conform modelului proiectului

În procesul de modelare, configurațiile inițiale ale componentelor sunt stocate în fișiere JSON de definiție afiliate componentei. Acestea sunt utilizate la fel pentru reprezentarea grafică în cadrul secțiunilor programului de modelare propus. Configurația proiectului se realizează prin interconectarea componentelor în secțiunea de configurare. După ce s-a definirt modelului proiectului urmează a fi va selectată opțiunea de generare a configurației către un fișier de format JSON, Fig. 3.6.

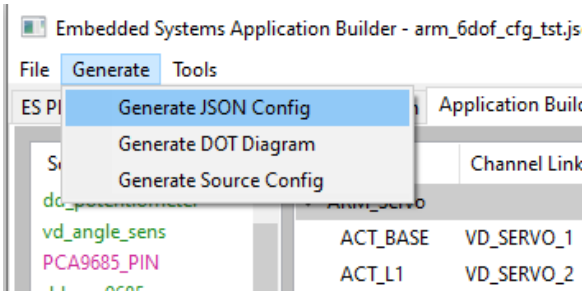


Fig. 3.6. Selectarea opțiunii de generare a configurației JSON a proiectului

O secvență a fișierului de configurație proiect este prezentată în Fig. 3.7.

Din secvența prezentată se poate vedea cum componenta "dd_potentiometer" interacționează cu componenta "mcal_adc" prin intermediul canalelor "Channels":{"POT_1":"ADC_1", "POT_2": "ADC_2", "POT_3": "ADC_3" }, definite în grupul "dd_pot", pe de o parte, și "mcal_adc", pe de altă parte, funcția de transfer utilizată pentru extragerea de informație fiind definită prin configurația "Pull": "MCAL_ADC_ReadChannel". Configurarea de interacțiuni în cadrul proiectului urmează principiul expus în cap 2. figura Fig. 2.40, pentru interacțiunea de componente.

```
{
    "Description": "Application Demo for 3-DOF Robotic arm  control",
    "git": "https://github.com/ML-ES-Platform/arm_6dof_demo.git",
    "Path": "ASW/arm_6dof_demo/",
    "Components": {
            "mcal_adc": {
            "git": "https://github.com/ML-ES-Platform/mcal_adc.git",
            "Path": "MCAL/mcal_adc/",
            "Groups": {
                "mcal_adc": {
                    "Channels": {
                        "ADC_1": "A3",
                        "ADC_2": "A4",
                        "ADC_3": "A5" }
                },
                "mcal_adc_pin": {
                    "Channels": {
                        "A3": "null",
                        "A4": "null",
                        "A5": "null" }
                }
            }
        },
        "dd_potentiometer": {
            "git": "https://github.com/ML-ES-Platform/dd_potentiometer.git",
            "Path": "ESW/dd_pot/",
            "Groups": {
                "dd_pot": {
                    "Channels": {
                        "POT_1": "ADC_1",
                        "POT_2": "ADC_2",
                        "POT_3": "ADC_3" },
                    "Dependency": "mcal_adc",
                    "Pull": "MCAL_ADC_ReadChannel" }
            }
        }
    }
}
```

Fig. 3.7. Secvență a fișierului de configurație proiect

Generarea arhitecturii conform modelului proiectului

Resursa de program dezvoltată permite vizualizarea grafică a componentelor și a interconexiunilor prin generare de diagrame arhitecturale aplatizate, similare cu cea prezentată în Fig. 3.8, în format Graphviz [134]. Această funcționalitate permite validarea că ce s-a pus ca scop corespunde cu ce s-a obținut în rezultat, la fel ca și o metodă de validare a metodei de modelare dezvoltate în teză.

În consecință, configurațiile de proiect definite prin modelare cu JSON se vor converti în fișiere de tip DOT, care sunt convertite de către instrumentul Graphviz într-o imagine tip diagramă de interconexiuni. Pe diagrama generată se poate vizualiza dacă canalele de transfer a informației sunt grupate corect, interconectează componentele dorite.

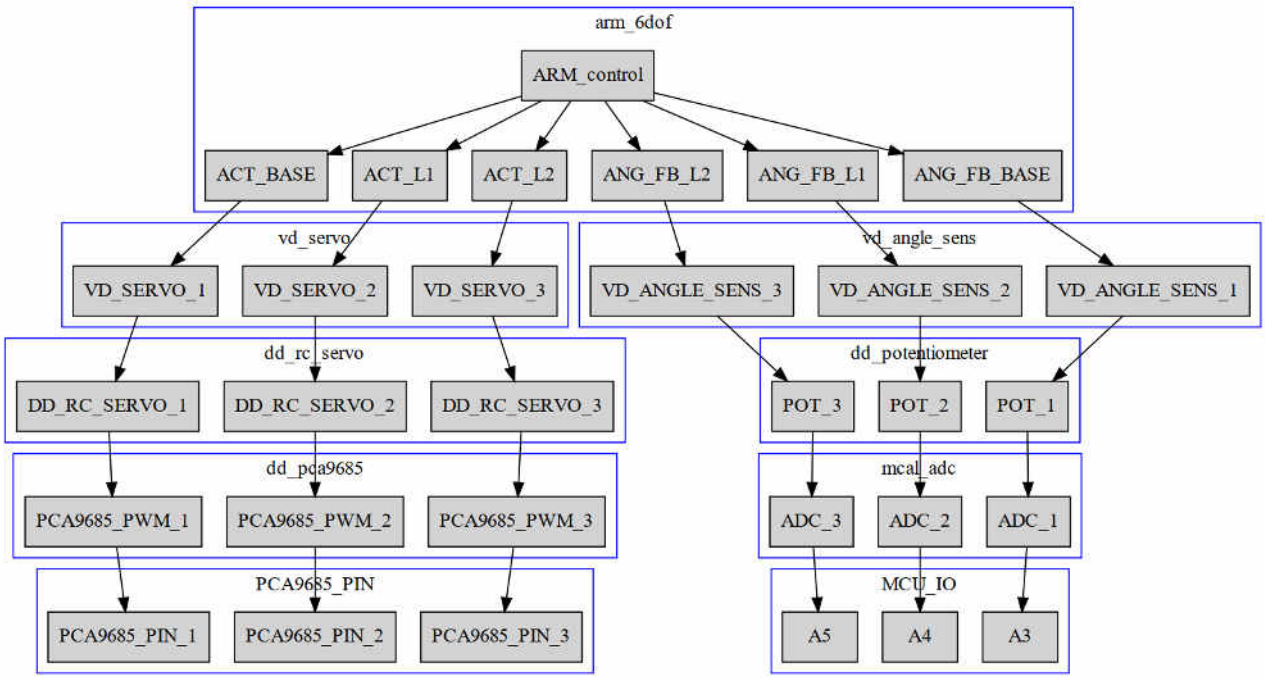


Fig. 3.8. Vizualizare grafică a interconexiunilor între canale [170]

Opțiunea se activează prin meniul resursei de program, după cum se vede în Fig. 3.9

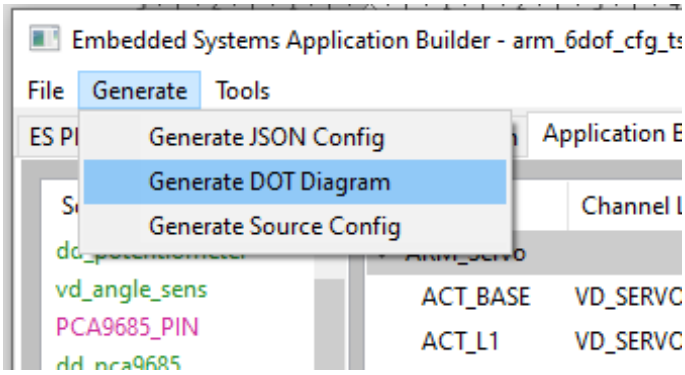


Fig. 3.9. Selectarea opțiunii de generare a diagramei arhitecturale DOT [170]

Un exemplu de conversie din configurație proiect JSON în configurație diagramă DOT este prezentata în Fig. 3.10.

Secvența de configurație JSON	Secvența de diagramă DOT

```
{
  "Description": "Application
Demo for 3-DOF Robotic arm
control",
  "git":
"https://github.com/ML-ES-
Platform/arm_6dof_demo.git",
  "Path": "ASW/arm_6dof_demo/",
  "Components": {
      "mcal_adc": {
      "git":
"https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Groups": {
        "mcal_adc": {
          "Channels": {
            "ADC_1": "A3",
            "ADC_2": "A4",
            "ADC_3": "A5" }
        },
        "mcal_adc_pin": {
          "Channels": {
            "A3": "null",
            "A4": "null",
            "A5": "null" }
        }
      }
    },
    "dd_potentiometer": {
      "git":
"https://github.com/ML-ES-
Platform/dd_potentiometer.git",
      "Path": "ESW/dd_pot/",
      "Groups": {
        "dd_pot": {
          "Channels": {
            "POT_1": "ADC_1",
            "POT_2": "ADC_2",
            "POT_3": "ADC_3" },
          "Dependency":
"mcal_adc",
          "Pull":
"MCAL_ADC_ReadChannel" }
      }
    }
  }
}
```

```
digraph {
  subgraph cluster_MCU_IO {
    node[shape=box
style=filled]
    color=blue
    label=MCU_IO
    A3
  }
  subgraph cluster_MCU_IO {
    node [shape=box
style=filled]
    color=blue
    label=MCU_IO
    A4
  }
  subgraph cluster_MCU_IO {
    node [shape=box
style=filled]
    color=blue
    label=MCU_IO
    A5
  }
  subgraph cluster_mcal_adc
{
    node [shape=box
style=filled]
    color=blue
    label=mcal_adc
    ADC_1
  }
  ADC_1 -> A3
  subgraph cluster_mcal_adc
{
    node [shape=box
style=filled]
    color=blue
    label=mcal_adc
    ADC_2
  }
  ADC_2 -> A4
  subgraph cluster_mcal_adc
{
    node [shape=box
style=filled]
    color=blue
    label=mcal_adc
    ADC_3
  }
  ADC_3 -> A5
  subgraph
cluster_mcal_adc {
    node [shape=box
style=filled]
    color=blue
    label=mcal_adc
    A3
  }
  subgraph
cluster_mcal_adc {
    node [shape=box
style=filled]
    color=blue
    label=mcal_adc
    A4
  }
  subgraph
cluster_mcal_adc {
    node [shape=box
style=filled]
    color=blue
    label=mcal_adc
    A5
  }
  subgraph
cluster_dd_potentiometer
{
    node [shape=box
style=filled]
    color=blue

label=dd_potentiometer
    POT_1
  }
  POT_1 -> ADC_1
  subgraph
cluster_dd_potentiometer
{
    node [shape=box
style=filled]
    color=blue

label=dd_potentiometer
    POT_2
  }
  POT_2 -> ADC_2
  subgraph
cluster_dd_potentiometer
{
    node [shape=box
style=filled]
    color=blue

label=dd_potentiometer
    POT_3
  }
}
```

Fig. 3.10. Exemplu de conversie JSON în DOT

Textul generat în format DOT se va converti într-o diagramă arhitecturală, după cum urmează din Fig. 3.11.

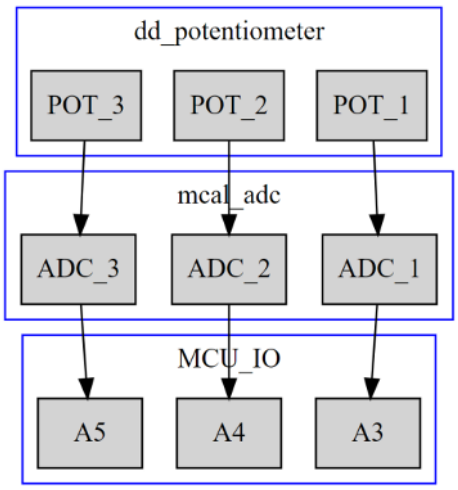


Fig. 3.11. Exemplu de diagramă arhitecturală generată [170]

Generarea resurselor de proiect conform modelului definit

Odată definit sistemul și validat din punctul de vedere al arhitecturii și interacțiunilor dintre componente, resursa de program elaborată are opțiunea de a genera fișiere de configurații cod sursă, care, fiind incluse în componentele din proiect, o adaptează spre utilizare în proiect conform configurației predefinite în procesul de modelare, funcționalitate accesibilă din meniul de generare, vezi Fig. 3.12.

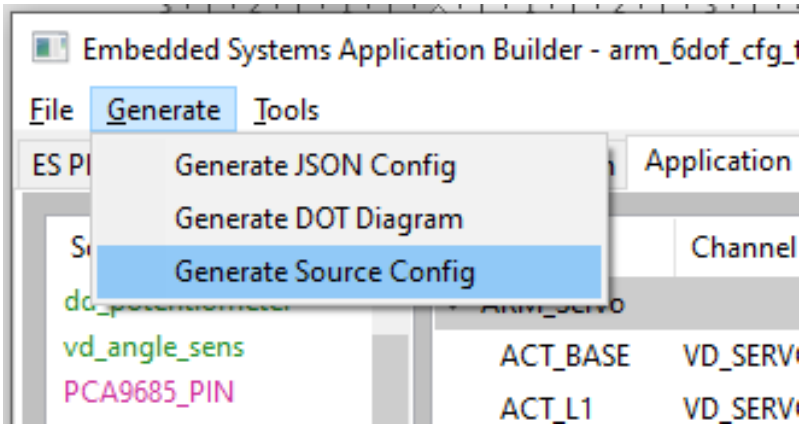


Fig. 3.12. Selectarea opțiunii de generare a configurațiilor cod sursă

Procesul de generare a configurației cod sursă va genera o pereche de fișiere – declarații și definiții ale proiectului, care prin ierarhia de incluziuni este preferențiat de toate componentele implicând configurarea lor. În continuare se prezintă un exemplu de fișiere generate pentru exemplul propus, Fig. 3.11. Pentru declarații vom avea codul sursă generat, prezentat în Fig. 3.13, iar pentru definiții vom avea codul sursă generat prezentat în Fig. 3.14.

```
/**
 * @file arm_6dof_demo_config.h
 */
#ifndef _ARM_6DOF_DEMO_CONFIG_H_
#define _ARM_6DOF_DEMO_CONFIG_H_
#include "platform_config.h"
#ifndef MCAL_ADC_CONFIG
#define MCAL_ADC_CONFIG
enum MCAL_ADC_IdType {ADC_1,ADC_2, ADC_3, MCAL_ADC_CHANNEL_NR_OF};
#include "MCAL/mcal_adc/mcal_adc.h"
#endif
#ifndef DD_POT_CONFIG
#define DD_POT_CONFIG
enum DD_POT_IdType {DD_POT_1,DD_POT_2, DD_POT_3, DD_POT_CHANNEL_NR_OF};
#include "ESW/dd_pot/dd_pot.h"
#endif
Std_ReturnType arm_6dof_demo_config(void);

#endif
```

Fig. 3.13. Codul sursă generat pentru declarații

```
/**
 * @file arm_6dof_demo_config.cpp
 */
#include "arm_6dof_demo_config.h"

Std_ChannelIdType ADC_PIN_Group[MCAL_ADC_CHANNEL_NR_OF] = {A3, A4, A5};
Std_ChannelIdType ADC_Group[MCAL_ADC_CHANNEL_NR_OF] = {MCAL_ADC_1, MCAL_ADC_2,
MCAL_ADC_3};
Std_ChannelIdType DD_POT_Group[DD_POT_CHANNEL_NR_OF] = {DD_POT_1, DD_POT_2,
DD_POT_3};

Std_ReturnType arm_6dof_demo_config(void)
{
    Serial.begin(115200);
    Serial.println("Application Demo for 3-DOF Robotic arm  control");
    Std_ReturnType error = E_OK;
  // Interconectare canale grupate
    error += MCAL_ADC_GroupSetup(ADC_Group, ADC_PIN_Group, MCAL_ADC_CHANNEL_NR_OF);
    Serial.print("MCAL ADC configured - Error : ");
    Serial.println(error);
  // Interconectare canale individual
    error += DD_POT_ChannelSetup(DD_POT_1, MCAL_ADC_1);
    error += DD_POT_ChannelSetup(DD_POT_2, MCAL_ADC_2);
    error += DD_POT_ChannelSetup(DD_POT_3, MCAL_ADC_3);

// Atribuire funcție de transfer către un grup (alternativ se poate individual
    error += DD_POT_SetGroupRawGetter(DD_POT_Group, MCAL_ADC_ReadChannel,
DD_POT_CHANNEL_NR_OF);
// ca alternativă se poate atribui individual per canal
// error += DD_POT_SetPullMethod(DD_POT_1, MCAL_ADC_ReadChannel);

    Serial.print("DD POT configured - Error : ");
    Serial.println(error);
    return error;
}
```

Fig. 3.14. Codul sursă generat pentru definiții

Urmând conceptul descris în capitolul 2, Fig. 2.42, pentru adaptarea componentei la platformă, aceasta va trebui sa conțină anumite funcționalități generale specifice platformei, pe lângă funcționalitățile specifice componentei. Aceste funcționalități specifice platformei pot fi parte de componentă când urmează procesul de dezvoltare din teză sau separat pentru cazurile în

care se reutilizează componente din alte surse externe, de exemplu, componentele din comunitatea Arduino. Un exemplu de funcționalități specifice platformei se poate vedea în Fig. 3.15 pentru declarații și Fig. 3.16 pentru definiții.

```
/*
 * mcal_adc.h
 *  Created on: May 8, 2020
 *      Author: Andrei Bragarenco
 */

#ifndef _MCAL_ADC_H_
#define _MCAL_ADC_H_

#include "mcal_adc_cfg.h"
#ifndef MCAL_ADC_CONFIG
enum MCAL_ADC_IdType { MCAL_ADC_CHANNEL_NR_OF = 0};
#endif

typedef struct MCAL_ADC_ChannelType_t{
    Std_ChannelIdType rawChannelId = 0;
    Std_RawGetterType GetRaw = NULL;
}  MCAL_ADC_ChannelType;

Std_ReturnType MCAL_ADC_ChannelSetup(Std_ChannelIdType channelId, Std_ChannelIdType
rawChannelId);
Std_ReturnType MCAL_ADC_GroupSetup(Std_ChannelIdType *srcIds, Std_ChannelIdType
*targhetIds, uint8_t nr_of_channels);

MCAL_ADC_ChannelType * MCAL_ADC_GetChannelRef(Std_ChannelIdType channelId);
Std_RawDataType MCAL_ADC_ReadChannelByRef(MCAL_ADC_ChannelType * channelRef);
Std_ReturnType MCAL_ADC_SetPullMethod(Std_ChannelIdType channelId, Std_RawGetterType
GetRaw);
Std_ReturnType MCAL_ADC_SetGroupRawGetter(Std_ChannelIdType *srcIdGroup,
Std_RawGetterType GetRaw, uint8_t nr_of_channels);
Std_RawDataType MCAL_ADC_ReadChannel(Std_ChannelIdType channelId);

#endif /* _MCAL_ADC_H_ */
```

Fig. 3.15. Codul sursă declarații funcționalități specifice platformei

```
/*
 * mcal_adc.cpp
 *  Created on: May 8, 2020
 *      Author: Andrei Bragarenco
 */

#include "mcal_adc.h"

#ifdef PLATFORM_CONFIG_ENABLE
extern MCAL_ADC_ChannelType MCAL_ADC_Channels[MCAL_ADC_CHANNEL_NR_OF];
#else
MCAL_ADC_ChannelType MCAL_ADC_Channels[MCAL_ADC_CHANNEL_NR_OF];
#endif
// extragerea referințe de canal din ID-ul sau
MCAL_ADC_ChannelType *MCAL_ADC_GetChannelRef(Std_ChannelIdType channelId) {
    MCAL_ADC_ChannelType *channelRef;

    if (channelId < MCAL_ADC_CHANNEL_NR_OF)  {
        channelRef = &MCAL_ADC_Channels[channelId];
    }
    else  {
        channelRef = NULL;
    }
    return channelRef;
}
// configurarea legăturilor de canale individuale
```

```
Std_ReturnType MCAL_ADC_ChannelSetup(Std_ChannelIdType channelId, Std_ChannelIdType
rawChannelId)
{
    Std_ReturnType error = E_OK;
    MCAL_ADC_ChannelType *channelRef = MCAL_ADC_GetChannelRef(channelId);

    if (channelRef != NULL)  {
        channelRef->rawChannelId = rawChannelId;
        error = E_OK;
    }
    else {
        error = E_NOT_OK;
    }
    return error;
}
// configurarea legăturilor de canale grupate
Std_ReturnType MCAL_ADC_GroupSetup(Std_ChannelIdType *srcIds, Std_ChannelIdType
*targhetIds, uint8_t nr_of_channels) {
    Std_ReturnType error = E_OK;

    for (size_t i = 0; i < nr_of_channels; i++) {
        Std_ChannelIdType srcId = srcIds[i];
        Std_ChannelIdType targhetId = targhetIds[i];
        error += MCAL_ADC_ChannelSetup(srcId, targhetId);
    }
    return error;
}
// Funcţie de transfer pentru extragerea datelor prin referinţă la canal
Std_RawDataType MCAL_ADC_ReadChannelByRef(MCAL_ADC_ChannelType *channelRef){
    Std_RawDataType adcValue = -1;
    if (channelRef != NULL)    {
#ifdef ARDUINO
        adcValue = analogRead(channelRef->rawChannelId);
#elif defined ESP
#endif
    }
    return adcValue;
}
// metoda de legare funcţie de transfer  de extragere Info de la componenta
referenţiata
Std_ReturnType MCAL_ADC_SetPullMethod(Std_ChannelIdType channelId,
                                      Std_RawGetterType GetRaw){
    Std_ReturnType error = E_OK;
    MCAL_ADC_ChannelType *channelRef = MCAL_ADC_GetChannelRef(channelId);

    if (channelRef != NULL)    {
        channelRef->GetRaw = GetRaw;
        error = E_OK;
    }
    else   {
        error = E_NOT_OK;
    }
    return error;
}
// metoda de legare funcţie de transfer de extragere pentru canale grupate
Std_ReturnType MCAL_ADC_SetGroupRawGetter(Std_ChannelIdType *srcIds,
Std_RawGetterType GetRaw, uint8_t nr_of_channels)
{
    Std_ReturnType error = E_OK;

    for (size_t i = 0; i < nr_of_channels; i++)    {
        Std_ChannelIdType srcId = srcIds[i];
        error += MCAL_ADC_SetPullMethod(srcId, GetRaw);
    }
    return error;
}
// Funcţie de transfer pentru extragerea datelor prin ID canal
Std_RawDataType MCAL_ADC_ReadChannel(Std_ChannelIdType channelId)
```

```
{
    Std_RawDataType adcValue = -1;
    MCAL_ADC_ChannelType *channelRef = MCAL_ADC_GetChannelRef(channelId);

    if (channelRef != NULL)      {
        adcValue = MCAL_ADC_ReadChannelByRef(channelRef);
    }
    Else {
        adcValue = -1;
    }
    return adcValue;
}
```

Fig. 3.16. Codul sursă definiții funcționalități specifice platformei

Adaptorul de platformă este modalitatea prin care interacționează toate componentele în conceptul definit în teză. Toate componentele trebuie să conțină în lista de funcționalități cele două mecanisme importante: *stabilirea de legături între canale* și *stabilirea funcției de transfer informație* de la o componentă la alta. Având în vedere că aceste funcționalități sunt similare pentru toate componentele, ele pot fi generate automat, denumite cu prefixe după denumirea de componentă și numele funcționalităților, si ca urmare să fie completată cu celelalte funcționalități specifice componentei în cauză.

Ca urmare a întregului proces de automatizare a modelării și generării de cod sursă de platformă, conform conceptului, setul minim de fișiere din care va fi constituită o componentă va putea arăta după cum urmează în Fig. 3.17.

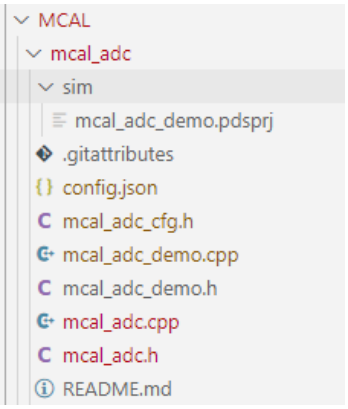


Fig. 3.17. Exemplu de colecție fișiere pentru o componentă

Fișierele din colecția tipică pentru o componentă, după exemplul din Fig. 3.17, vor avea următoarele destinații:

- sim/mcal_adc_demo.pdsprj – un exemplu de schemă electrică, cu posibilitate de simulare pentru demonstrarea funcționalității componentei.
- .gitattributes – fișier de configurații pentru platforma online unde este localizată componenta pentru reutilizare. Ar putea exista și alte fișiere, în funcție de modul în care se realizează accesul la platforma online.

- config.json – definiția componentei în format JSON care este utilizată pentru modelarea proiectelor prin metoda din teză.
- mcal_adc_cfg.h – configurația locală a componentei care este utilizată în cazul în care nu există o configurație globală în cadrul proiectului în care este utilizată.
- mcal_adc_demo.cpp – un proiect de demonstrație de utilizare a componentei, utilizat în combinație cu modelul de simulare.
- mcal_adc_demo.h – declarația funcțiilor principale de activare a proiectului de demonstrare care sunt invocate pentru a porni programul de demonstrație.
- mcal_adc.cpp – definițiile și implementările funcționalităților componentei.
- mcal_adc.h – declarațiile componentei sub formă de prototipuri de funcții, interfețe și structuri de date.
- README.md – informații generale pentru utilizatorii de componentă, o descriere succintă a funcționalităților și a modului de utilizare.

Reutilizarea componentelor de pe platforma online în cadrul modelului de proiect

O componentă, odată definită în cadrul oricărui proiect, conform conceptului din teză, este plasată pe platforma online împreună cu definiția acestuia în format JSON. Simultan, referința de componentă se va introduce în fișierul de listă de componente de pe platformă pentru a fi pusă la dispoziție utilizatorilor platformei. În procesul de modelare a următoarelor proiecte cu această platformă, o componentă selectată pentru a fi inclusă în model, se poate utiliza în proiect cu opțiunea *"Load Platform Components"* din Fig. 3.18. Această opțiune actualizează definițiile de componente ale platformei pe stația locală din mediul online, unde sunt definite și posibilele dependențe de alte componente. Acest lucru permite nu numai reutilizarea componentei, dar și experiența utilizării acesteia în diverse combinații, propunând recomandări de autocompletare expuse în secțiunea de modelare a proiectului. Noile dependențe și rafinamente ale definiției componentei se vor readuce către componentă prin configurațiile acestora, implicând stocare de experiență de utilizare a componentei.

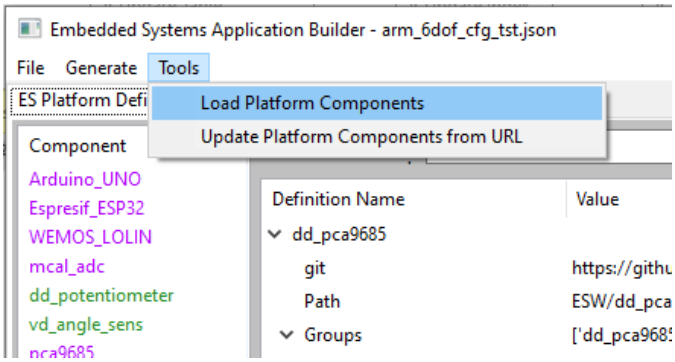


Fig. 3.18. Opțiunea de a descărca componenta de platformă selectată în proiect [170]

Aceasta modalitate de lucru iterativ cu componentele din platformă în diverse proiecte și în diverse configurații deschide noi oportunități de dezvoltare a metodologiei propuse în teză, în special cea de automatizare a modelării sistemelor electronice distribuite.

3.3 Rezultatele modelării sistemelor electronice distribuite

Metodologia de modelare a sistemelor electronice distribuite cu generarea automată a configurației propusa a fost valorificată prin dezvoltarea a diverse proiecte, cele mai importante fiind prezentate în subsecțiunile următoare. Fiecare dintre cele patru aplicații prezentate în subsecțiuni reprezintă validarea și testarea rezultatelor pe sisteme realizate în practica după cum urmează:

- *Sistemul de monitorizare a mediului* reprezintă validarea organizării componentelor generice cum ar fi senzor, interacțiunea cu utilizatorul și comunicare cu arhitecturi în straturi pentru abstractizarea MCU – microcontroler, ECU – componente electronice și SRV – servicii. La fel a fost realizat un servicii de senzori virtuali realizați printr-un model matematic și prin acces prin rețele de comunicare tip IoT [155].

-*Sistemul de control al brațului robotic* reprezintă validarea organizării componentelor tip senzor-actuator ca si componente cu lanțuri similare dar inverse de comunicare. Totodată prin acest proiect s-a validat lanțul universal de comunicare în sistemele electronice distribuite care asigura conexiunea unui senzor de pe un dispozitiv la distantă ca și componenta al altui dispozitiv [168].

-*Sistemul de orientare a reflecției luminii* reprezintă validarea integrării componentelor realizate prin modelare matematica în sistemele electronice distribuite interconectate cu componentele sistemului utilizând metodologia de lanțuri de comunicare propusă în teză de către autor.

-*Sistemul de control al camerei de uscare de fructe* reprezintă o implementare în cadrul unui proiect instituțional a unui sistem de control a unei instalații componente de reglarea automată si automate finite, prin utilizarea conceptelor de organizare a sistemelor electronice distribuite propuse în teză.

În secțiunile ce urmează se prezinta aplicațiile practice prin care au post testate și validate conceptele propuse de autor în teză, cu descriere detaliata a sistemelor la nivel de sistem, modelarea componentelor prin arhitecturi în straturi însoțite de prezentară grafica a configurațiilor generate în urma utilizării produsului program dezvoltat pentru validarea metodologiei, si prezentat în secțiunea 3.2. Configurațiile JSON generate prin intermediul produsului program menționat conform metodologiei propuse sunt prezentate în Anexa 4. Configurațiile de proiect

pentru una dintre aplicațiile practice – *Sistem de control al brațelor robotice,* în format cod sursa C++ generate cu produsul program conform configurațiilor generate se regăsește în Anexa 5. iar codul sursa pentru unele componente generate – în Anexa 6.

3.3.1 Sistem de monitorizare a mediului

În baza cercetărilor efectuate, a fost proiectat un sistem pentru monitorizarea mediului înconjurător, în funcție de parametrii de temperatură, umiditate, detectarea prezenței, luminozitatea, zgomotul, detectarea CO2. Pentru definiția arhitecturii sistemului, a fost utilizat conceptul unei *arhitecturi în straturi* organizate în stive. Acest lucru permite o mai bună organizare a proiectului, precum și reutilizarea și extensibilitatea componentelor sistemului.

Conceptul de sistem

Scopul sistemului este de a colecta date de mediu de pe dispozitive situate la diferite coordonate geografice. Datele vor fi stocate pe un server dedicat sistemului, cu posibilitatea de a vizualiza datele atât afișate din componenta dispozitivului server, cât și prin Internet, de pe o pagină web, găzduită de serverul dedicat. Arhitectura la nivel de sistem este realizata prin diagrama conceptuală, prezentată în Fig. 3.19.

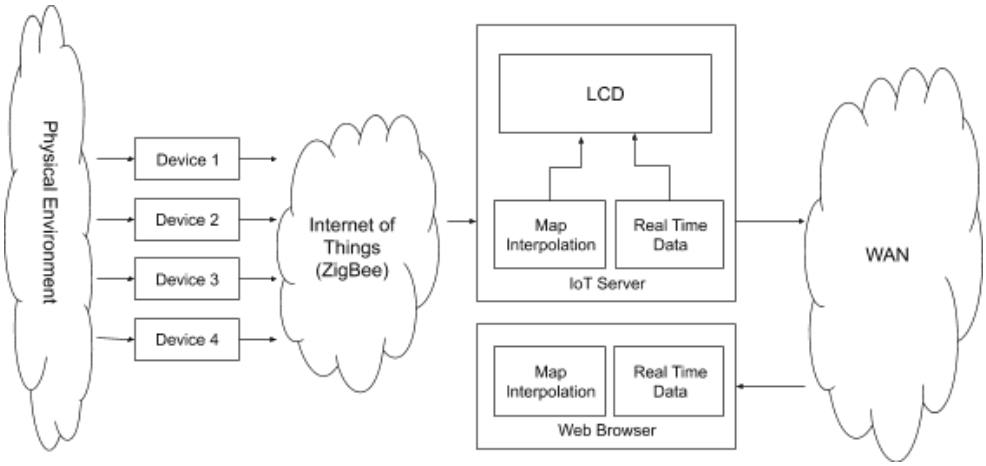


Fig. 3.19. Arhitectura generală a sistemului [155]

Fiecare dintre dispozitive de achiziție din sistemul menționat este responsabil pentru colectarea datelor în zona în care este amplasat, iar datele vor fi prelucrate de alte sisteme pentru care aceste date prezintă interes. În acest sistem, datele sunt colectate pentru a realiza o hartă climatică pe diverși parametri. Harta este construită prin metoda descrisă în teză, în capitolul 2 și permite estimarea valorii parametrului în zonele care nu sunt acoperite cu dispozitive de achiziții. Adițional, este posibil ca aceste date să poată fi utilizate în alte scopuri, cum ar fi reacționarea și o eventuală intervenție în anumite situații în zona de interes.

Dispozitive de achiziție

Conform arhitecturii sistemului, dispozitivele au fost proiectate în conformitate cu principiile descrise în metodologia propusă în teză, capitolul 2. Structural, fiecare componentă a senzorului sau a comunicării este realizată conform conceptului de componentă generică prezentat în teză. Respectiv, fiecare dintre componente are o stivă împărțită în straturi, care oferă nivelului de abstracție al aplicației numeroase servicii prin interfața RTE. În ansamblu, dispozitivul are structura prezentată în Fig. 3.20.

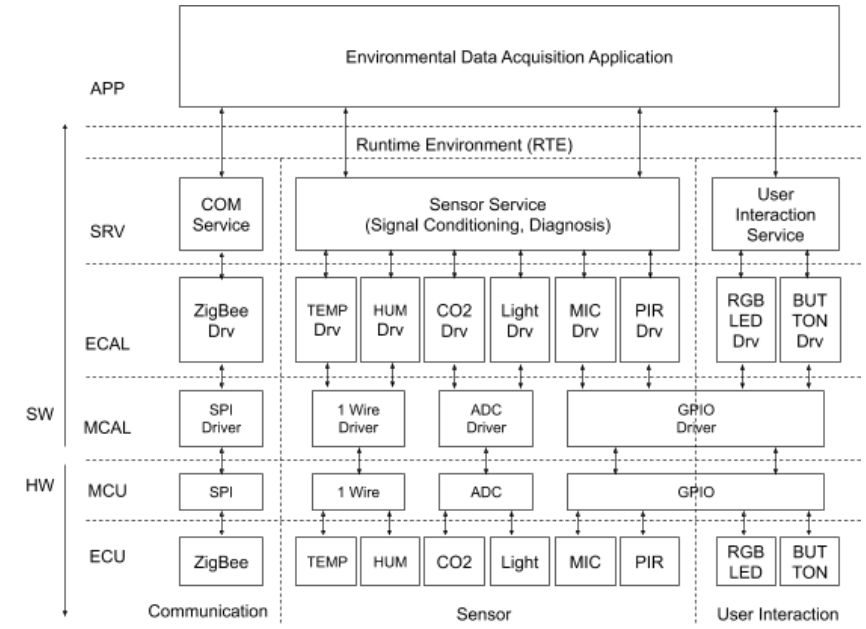


Fig. 3.20. Arhitectura în straturi a dispozitivului IoT [155]

Utilizând instrumentul de configurare și generare prezentat în teză, obținem o *arhitectură în straturi* pentru componentele de platformă, după cum e prezentat în **Fig. 3.21**.

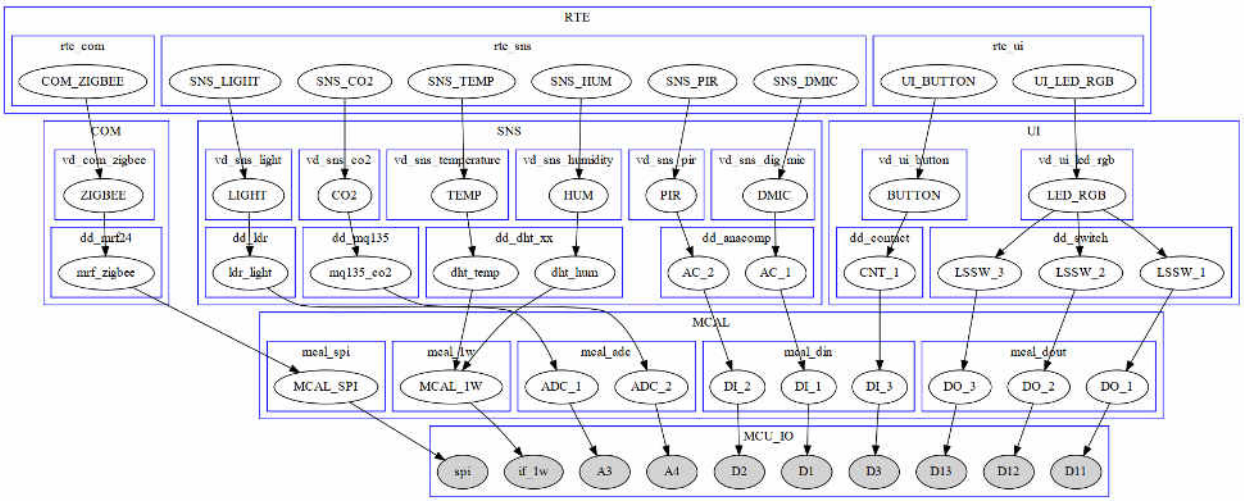


Fig. 3.21. Arhitectura în straturi a dispozitivului IoT generată conform configurației

Funcțional, dispozitivele colectează date despre mediu și le transmit rețelei IoT. Pe partea de control, se realizează funcționalități minime, diagnosticele de mediu sunt semnalizate secvențial

către un LED RGB, unde fiecare dintre cele opt culori primare indică un diagnostic de prag pe un parametru specific. Butonul de pe dispozitiv are funcția de a trimite un semnal de identificare, utilizat pentru a configura sistemul. Fluxul de date și semnalele de control sunt prezentate în diagrama funcțională din Fig. 3.22.

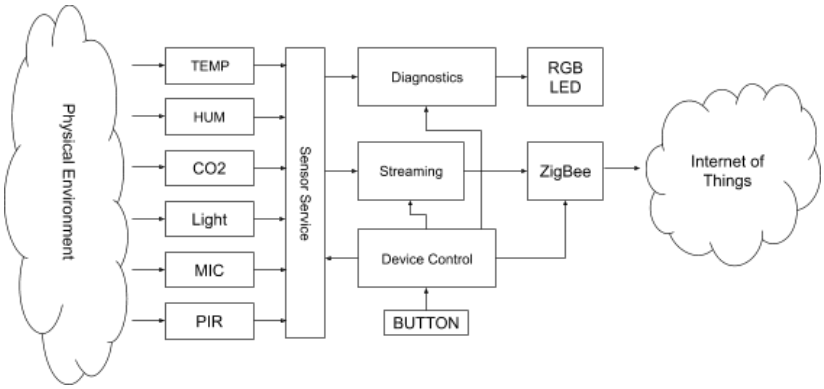


Fig. 3.22. Diagrama funcțională și de flux de date pentru dispozitivul IoT [155]

Pentru componentele de la nivelul de aplicație al dispozitivului de senzori IoT vom avea o arhitectură de sistem generată, după cum urmează în Fig. 3.23. Arhitectura sistemului pentru dispozitivul IoT generată conform configurației.

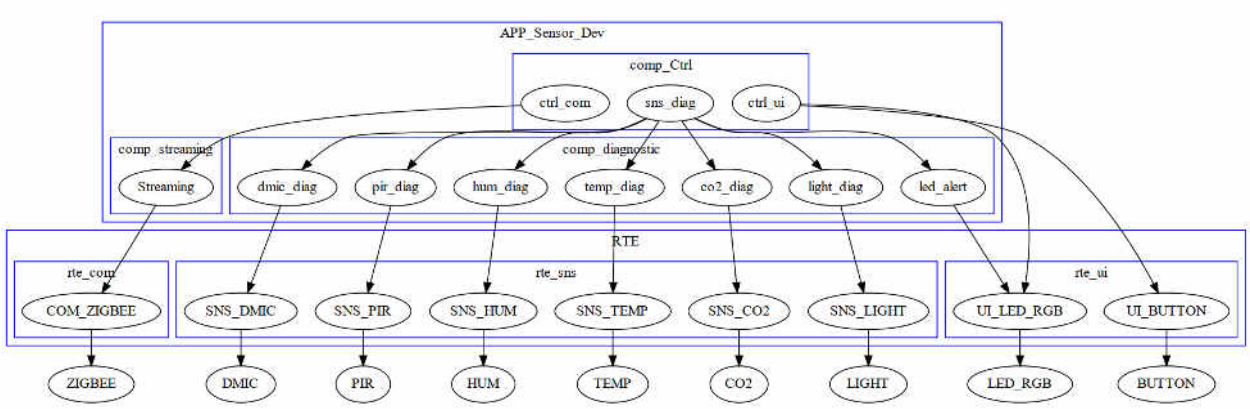


Fig. 3.23. Arhitectura sistemului pentru dispozitivul IoT generată conform configurației

Dispozitivele din cadrul sistemului sunt construite luând în considerație constrângerile de costuri reduse de realizare în producție. Din punct de vedere electric, dispozitivele conțin o colecție minimă de componente – senzori, microcontroler ieftin, modul de comunicare ZigBee, buton, led RGB. Aceste constrângeri necesită ca funcțiile de condiționare să fie efectuate în domeniul software. Pentru demonstrații, a fost construit un prototip de dispozitiv (Fig. 3.24) prezentat de către autor în lucrarea [155].

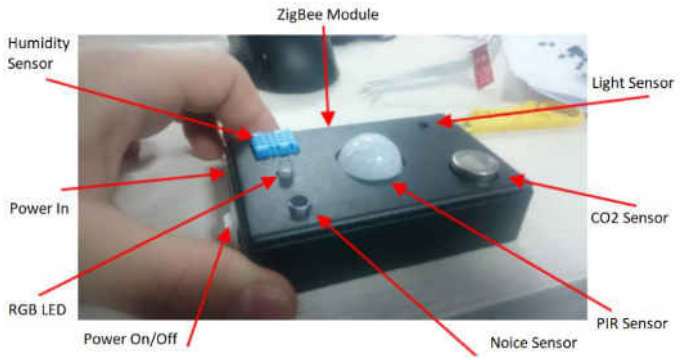


Fig. 3.24. Prototipul dispozitivului IoT pentru achiziționarea datelor de mediu

Coordonator server de achiziție

În implementarea prezentată în teză, serverul de achiziție prin rețeaua Zigbee funcționează ca un coordonator ZigBee (ZC), nodul central din rețea, de la care sunt accesibile toate dispozitivele conectate la rețea. Structural, implementarea este un sistem complet, în special din partea conectivității senzorilor datorită conceptului arhitectural, unde serviciile senzorilor abstractizează componentele senzorului. În cazul serverului de achiziție, fluxul de date de la senzori este direcționat prin modulul de comunicație și, respectiv, solicitările de date de la senzori sunt înlocuite cu solicitări către rețea. Din punctul de vedere al stratului de aplicație, accesul la date din stratul de servicii prin RTE, senzorii sunt văzuți ca și cum ar fi parte a echipamentului dat, cu fluxul de informație direcționat prin componente de comunicare.

În Fig. 3.25 este prezentată *arhitectura în straturi* a sistemului serverului de coordonare, inclusiv reprezentarea stratului inferior al stivei de componentelor de senzori, împreună cu legătura dintre serviciile senzorilor și serviciile de comunicații.

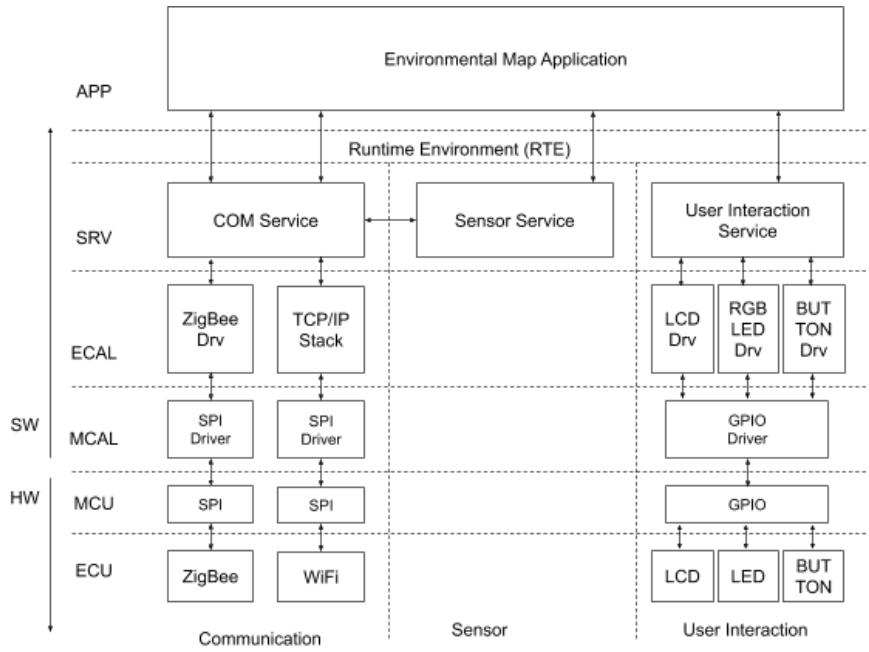


Fig. 3.25. Arhitectura în straturi a serverului IoT

Prin utilizarea instrumentului de configurare și generare prezentat în teză obținem o *arhitectură în straturi* a serverului IoT pentru format din nivelele de platformă, după cum e prezentat în Fig. 3.26.

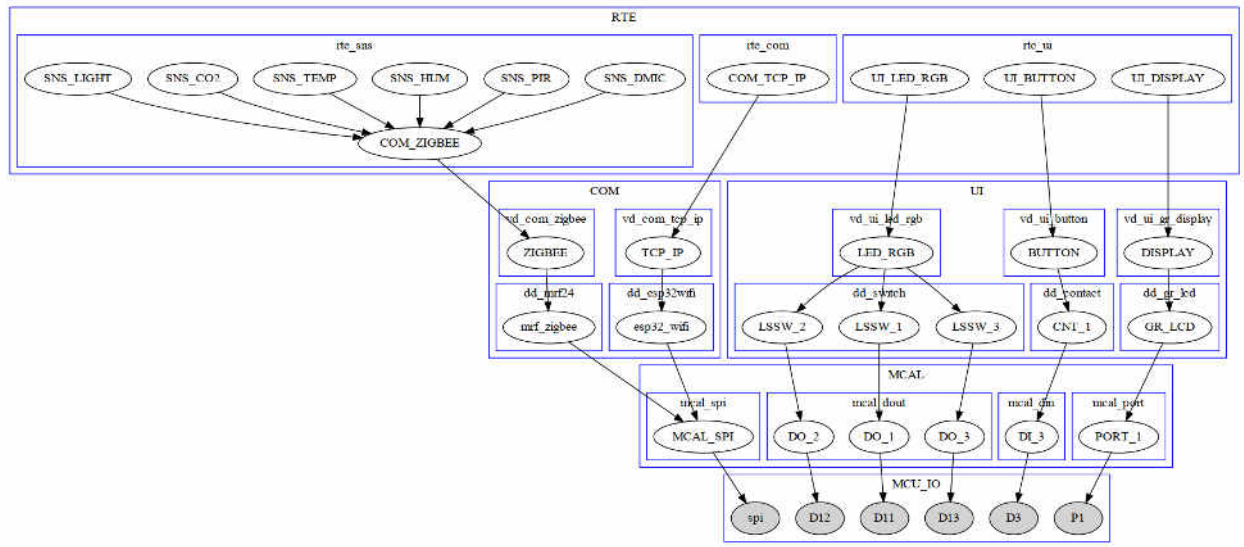


Fig. 3.26. Arhitectura în straturi a serverului IoT generată conform configurației

O altă caracteristică a serverului pentru achiziționarea de date de mediu prin rețeaua ZigBee este aceea că poate comunica cu rețeaua de Internet obișnuită prin rețeaua WiFi. În arhitectură, se poate observa că stiva de comunicații TCP / IP cu toate componentele pentru accesul LAN / WAN este inclusă printre componentele de comunicație.

Funcțional, sistemul de coordonare, prezentat în Fig. 3.27, reprezintă un flux de informații din rețeaua dispozitivelor interconectate prin tehnologia ZigBee către rețeaua globală. Sistemul implementează local funcționalități pentru reprezentarea datelor într-o hartă de interpolare și monitorizarea în timp real a senzorului selectat. Datorită achiziției senzorului în stratul de service al senzorului, aplicația folosește datele de la senzori ca și cum ar fi conectată direct la echipamentul pe care rulează. Acesta implementează conceptul de circuit electronic distribuit, care este propus în teză. Accesând rețeaua globală de Internet prin WiFi, serverul coordonator acționează ca un Gateway care conectează rețeaua IoT prin tehnologia ZigBee. Aceasta efectuează și serviciile de achiziție în afara rețelei de achiziții.

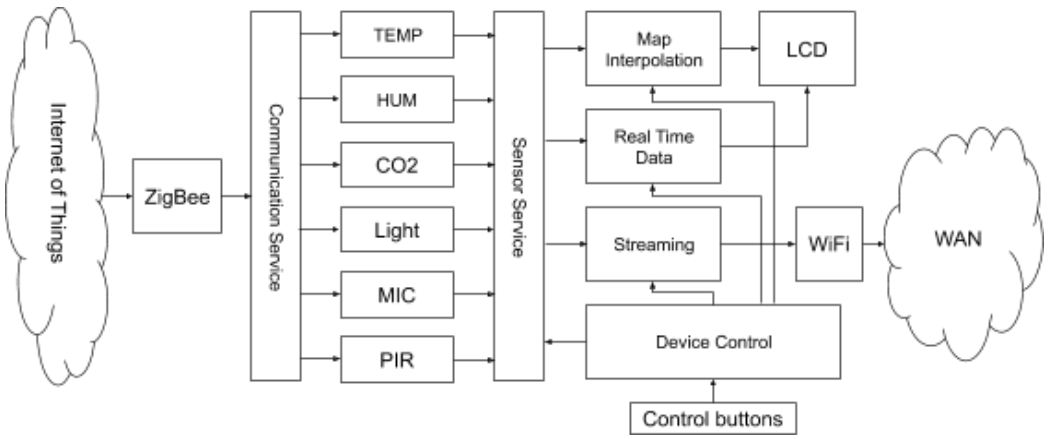


Fig. 3.27. Diagrama funcțională și de flux de date a coordonatorului serverului IoT [155]

Pentru componentele de la nivelul de aplicație vom avea o arhitectură de sistem generată, după cum urmează în Fig. 3.28.

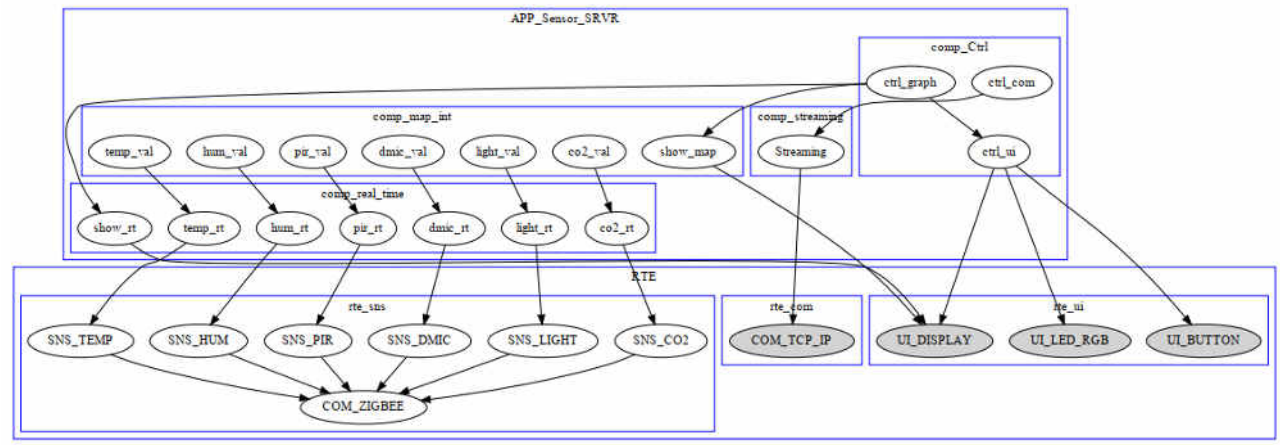


Fig. 3.28. Arhitectura sistemului pentru serverul IoT generată conform configurației

Implementarea prototipului pentru serverul de coordonare a fost realizată pe o placă de dezvoltare, care are la bord diferite tipuri de conexiuni wireless. Pe lângă faptul că procesorul are o performanță moderată, pe această platformă întreaga arhitectură menționată mai sus a fost realizată în conformitate cu principiile descrise în teză. Funcționalitățile interacțiunii cu utilizatorul, interpolarea hărții și a unui server Web, pentru a accesa datele din rețeaua globală, sunt realizate în cadrul plăcii de dezvoltare prezentate în Fig. 3.29.

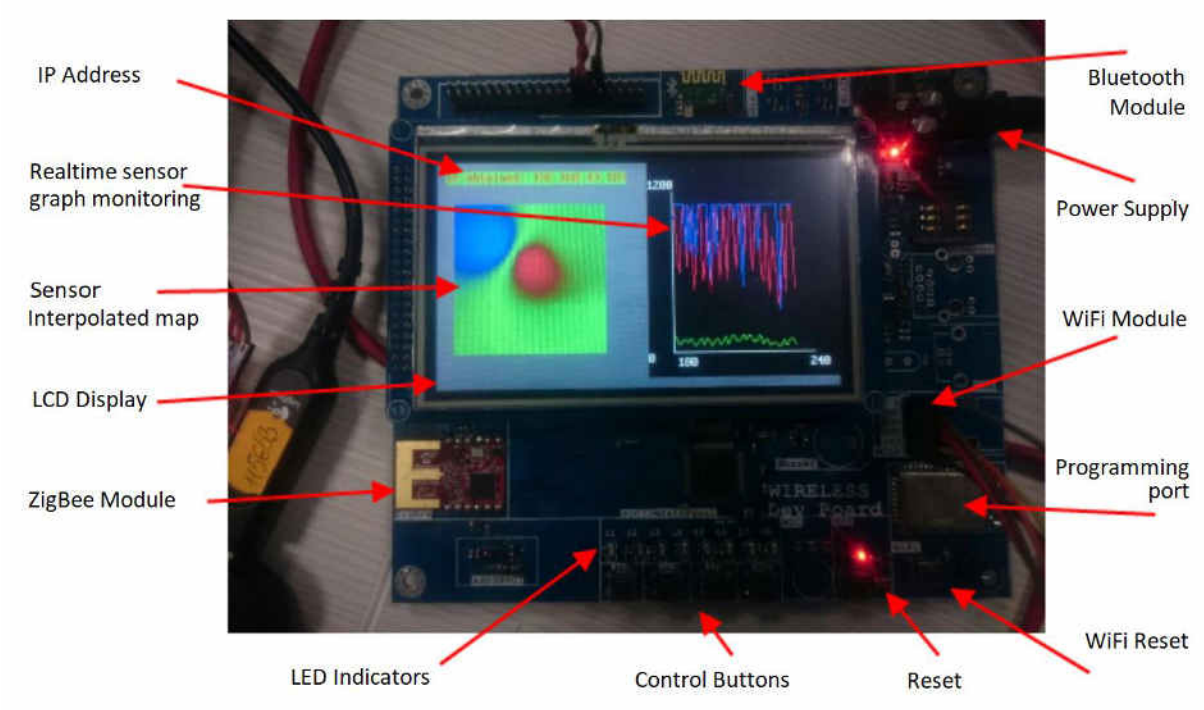


Fig. 3.29. Prototipul serverului IoT pentru achiziționarea datelor de mediu [178]

Serviciul web

Pentru versiunea de prototip, serviciul web de pe serverul de coordonare oferă o pagină simplă, duplicând funcționalitățile realizate în interacțiunea cu utilizatorul pe echipamentul care rulează serverul pentru selectarea parametrului vizualizat și monitorizarea în timp real a datelor din senzor, Fig. 3.30.

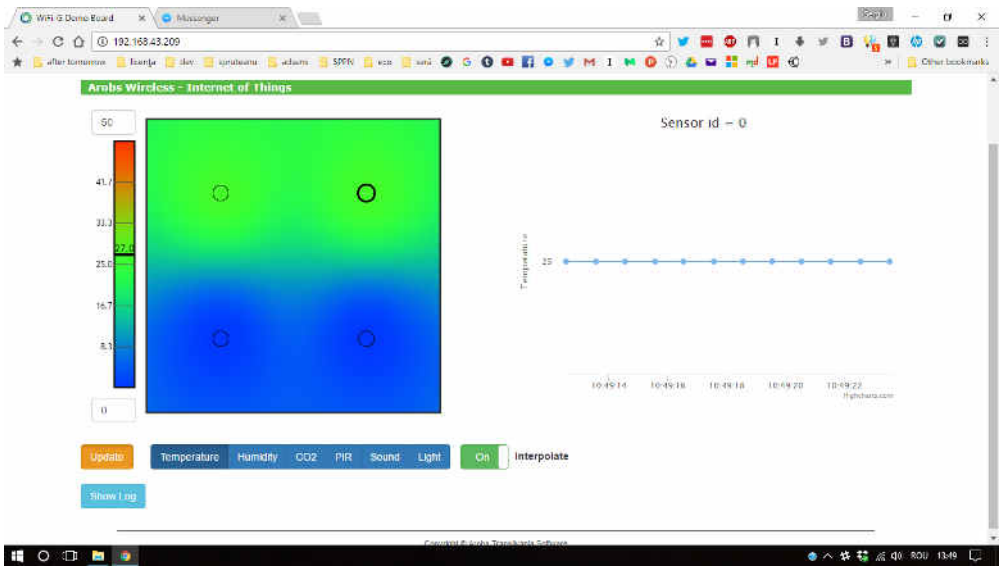


Fig. 3.30. Serviciul web pentru achiziționarea datelor de mediu [178]

Modul de operare

Modul de operare cu sistemul este unul simplu. Setarea constă în distribuirea dispozitivelor de achiziție în coordonatele de interes și înregistrarea coordonatelor în sistem pe dispozitiv ca parametru transmis și pe serverul de coordonare. Din interfața de utilizare a serverului de coordonare, stratul de parametri pentru vizualizarea pe hartă poate fi selectat prin butoane de control, precum și dispozitivul pentru vizualizarea datelor în timp real. Similar este modul de operare de pe pagina web furnizată de serverul Coordonator.

Aplicații și cazuri de utilizare

Domenii de aplicație tipice pentru utilizarea acestui sistem sunt monitorizarea unei zone în scopul monitorizării parametrilor mediului și detectarea situațiilor speciale. Fiecare dintre straturile de monitorizare oferă informații specifice parametrului pe care îl monitorizează. Ca aplicație tipică poate fi considerată securizarea unei încăperi. Cazurile de utilizare a sistemului oferă funcționalități, după cum urmează:

- Harta senzorului PIR va urmări accesul autorizat sau neautorizat într-o zonă desemnată, precum și pista în mișcare a unui obiect din zonă.
- Harta de monitorizare a temperaturii va furniza harta de distribuție termică, din care pot fi extrase informații despre scăderile de temperatură din cauza unei ferestre deschise sau semnalizarea temperaturii excesive din cauza unei surse de incendiu.
- Harta de luminozitate va oferi informații despre nivelul de lumină din cameră, care poate fi utilizat pentru a asigura normele de lucru de la punctul de deviere a cantității de lumină necesare activității umane, pentru a detecta defectele de iluminat, precum și pentru a identifica situațiile în care cineva a uitat să deconecteze lumina pentru economisirea energiei.
- Harta de zgomot va detecta dacă mediul respectă normele legale din zgomot, dar va detecta și incidente specifice, cum ar fi spargerea sticlei sau căderea obiectelor.
- Harta senzorului de CO2 va fi utilizat pentru detectarea și monitorizarea calității aerului în zona de interes, precum și pentru semnalizarea unor surse potențiale de foc sau fum excesiv, de exemplu, fumatul neautorizat în camere.
- Harta de umiditate poate fi utilizată pentru a monitoriza calitatea mediului, pentru a asigura standarde de lucru pentru personalul din încăpere.

Prin fuzionarea hârților menționate sistemul poate oferi diagnostice specifice și detecta situații mai complexe. De exemplu, un incendiu poate fi detectat de senzorii de CO2, luminozitate, temperatură și umiditate, o analiză mai atentă a acestor parametri ar putea oferi informații mai detaliate despre incident.

În cazul detectării anumitor nereguli, va fi generat un raport despre situația care va conține parametrul detectat, coordonatele și recomandări pentru remedierea problemei.

3.3.2 Sistem de control al brațelor robotizate

Poziționare braț robotizat 3-DOF

Ca urmare a metodologiei elaborate în teză, se propune o arhitectură pentru un sistem de control al brațelor robotizate. Local brațul robotizat 3-DOF poate fi controlat prin butoane și joystick sau prin comenzi printr-un terminal. Funcția terminalului ne permite să conectăm sistemul la un PC prin USB sau la un dispozitiv mobil prin Bluetooth. Ar putea accepta comenzi prin Internet prin intermediul serviciului de comunicații, astfel s-ar putea executa lucrări în colaborare, organizând o rețea de tip IIoT. Arhitectura sistemului poate fi văzută în Fig. 3.31.

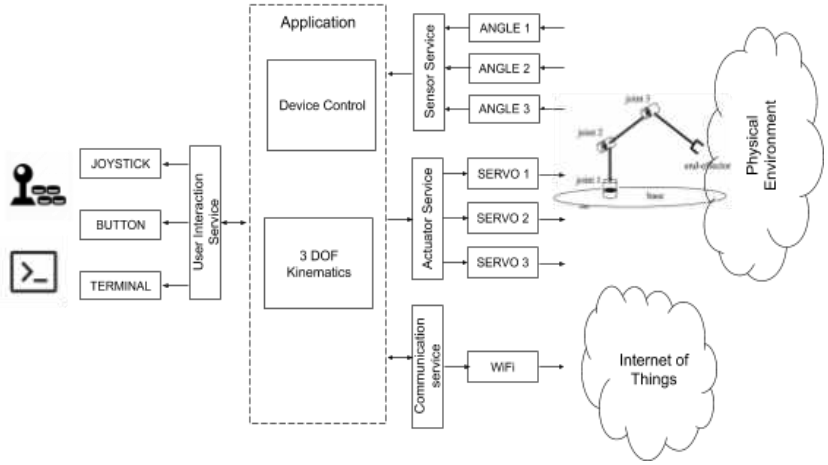


Fig. 3.31. Arhitectura de sistem a aplicației de control a brațului 3-DOF

Arhitectura sistemului la nivel de aplicație generată în urma proiectării cu instrumentul de configurare prezentat în teză va arăta ~~ca~~ după cum e prezentat în Fig. 3.32.

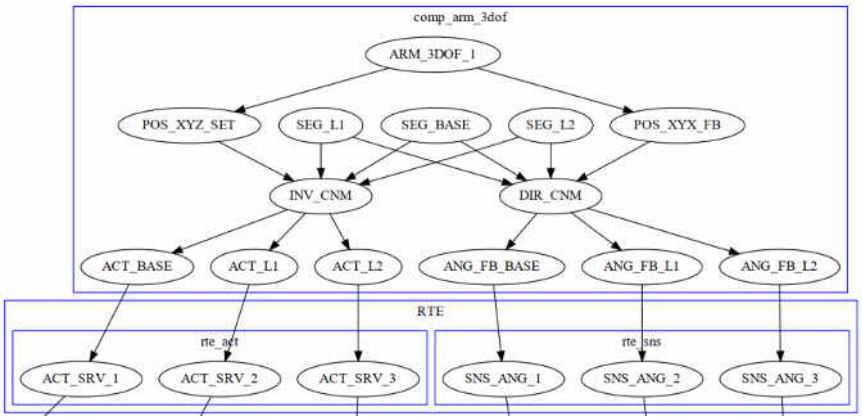


Fig. 3.32. Arhitectura de sistem al aplicației de control a brațului generată conform configurației

Poziția capului brațului robotizat este evaluată prin ecuații de cinematică inversă și validată cu evaluare cinematică directă, conform modelului descris în capitolul 2. Acest fapt permite ignorarea, evitarea sau protejarea de acțiuni mecanice în afara razei de acțiune sau pentru evitarea țintelor inaccesibile sau interzise. În Fig. 3.33, este prezentată diagrama conceptuală a protecției de limitare a mișcării brațului în afara zonei accesibile.

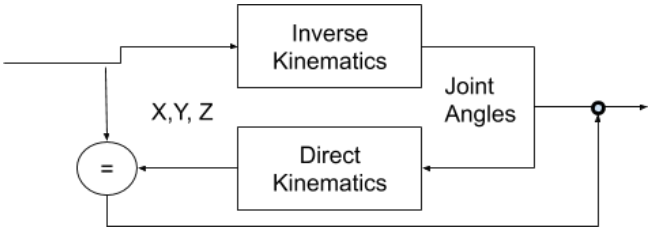


Fig. 3.33. Protecția out of range pentru aplicația de control a brațului [168]

Toate interacțiunile, cum ar fi interacțiunea cu utilizatorul, comunicarea în rețea și senzorul-actuator, sunt implementate urmând principiul de *arhitectură de componente în straturi* propusă în teză, vezi Fig. 3.34.

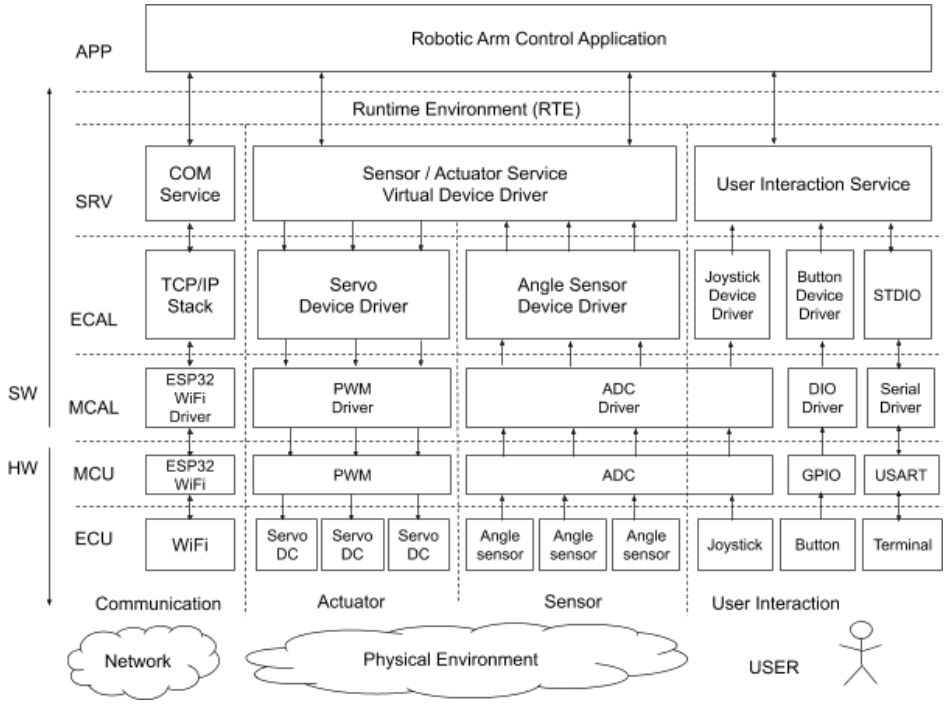


Fig. 3.34. Arhitectura în straturi a aplicației de control a brațului 3-DOF [168]

Configurațiile interconexiunilor pentru componentele de senzor-actuator stabilite cu instrumentul de configurare automată prezentat în teză vor arăta ca și în schița arhitecturală generată, prezentată în Fig. 3.35.

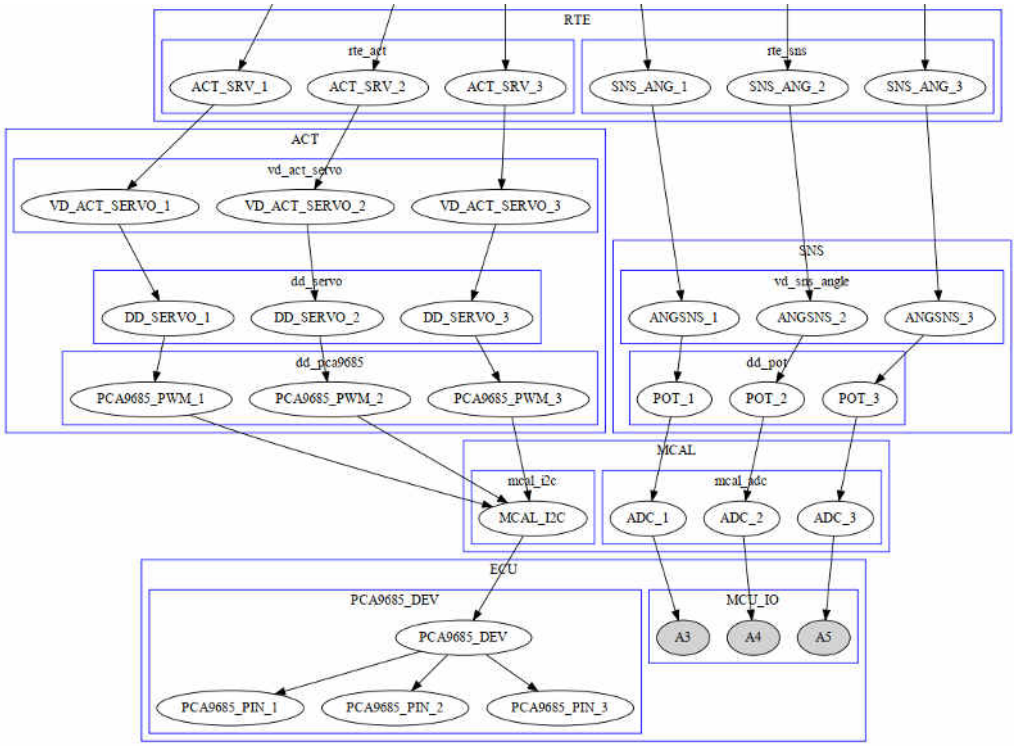


Fig. 3.35. Arhitectura în straturi a aplicației de control a brațului generată conform configurației

Modulul de control al aplicației acceptă comenzi de la interfețele de interacțiune a utilizatorului, joystick-ul, terminalul și butoane, prin Internet.

Braț robotic controlat la distanță cu un manipulator de referință

Urmând conceptul dispozitivelor electronice distribuite, discutat în teză, a fost dezvoltat un sistem mecatronic de manipulare la distanță. Aplicația este testată mai întâi pe un sistem de braț robot 3-DOF, urmând a fi extinsă la sisteme mecatronică complexe cu mai multe grade de libertate (Degree of Freedom – DOF). Întregul sistem este format din două manipulatoare, reprezentate de două brațe robotizate. Acestea sunt dotate cu servomotoare în calitate de actuatori și senzori de unghi pentru colectarea de răspuns a poziției curente a brațului. Brațele sunt conectate prin rețeaua fără fir, în cazul dat, WiFi. Unul dintre brațe nu folosește servomotoarele și este utilizat ca referință prin colectarea poziției articulațiilor brațului și a poziției capului efector. Celălalt braț, cu servomotoarele active, primește aceste informații și le interpretează ca un punct de referință pentru propriile articulații și poziția capului. Diagrama arhitecturii sistemului conceptual este prezentată în Fig. 3.36.

Conform abordărilor de *arhitectura în straturi* și *lanț de comunicare* propuse în teză, semnalul de la brațul de referință de la distanță, colectat de pe senzorii articulațiilor brațului de referință, traversează întregul *lanț de comunicare* și ajung la componentele de servicii ale brațului

actuator, cu servomotoarele active. Servomotoarele sunt poziționate conform unghiurilor colectate de la brațul de referință.

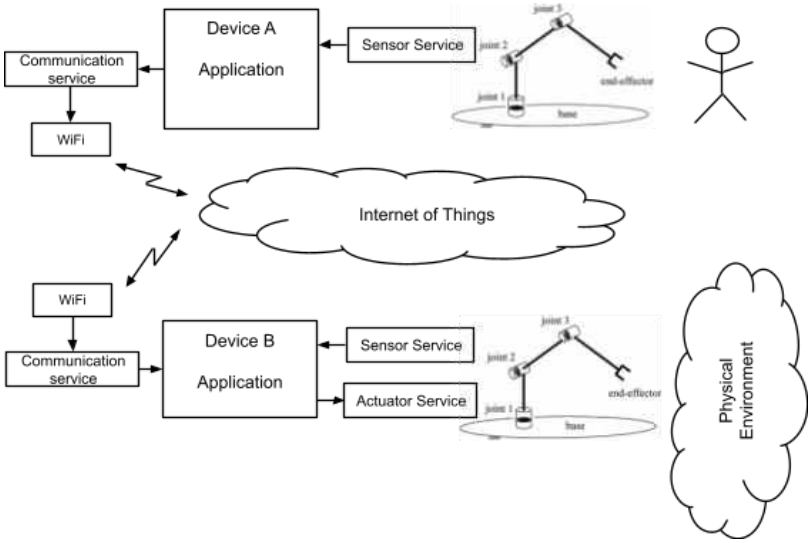


Fig. 3.36. Arhitectura sistemului pentru controlul manipulatorului de la distanță

Conform abordării prezentate, se abstractizează rețeaua și se consideră că senzorul de referință a poziției este conectat direct la dispozitivul de acționare. Diagrama din Fig. 3.37 reprezintă *arhitectura în straturi* a sistemului și fluxurile sau *lanțurile de comunicare* pentru semnalele achiziționate și semnalele de control pentru acționare.

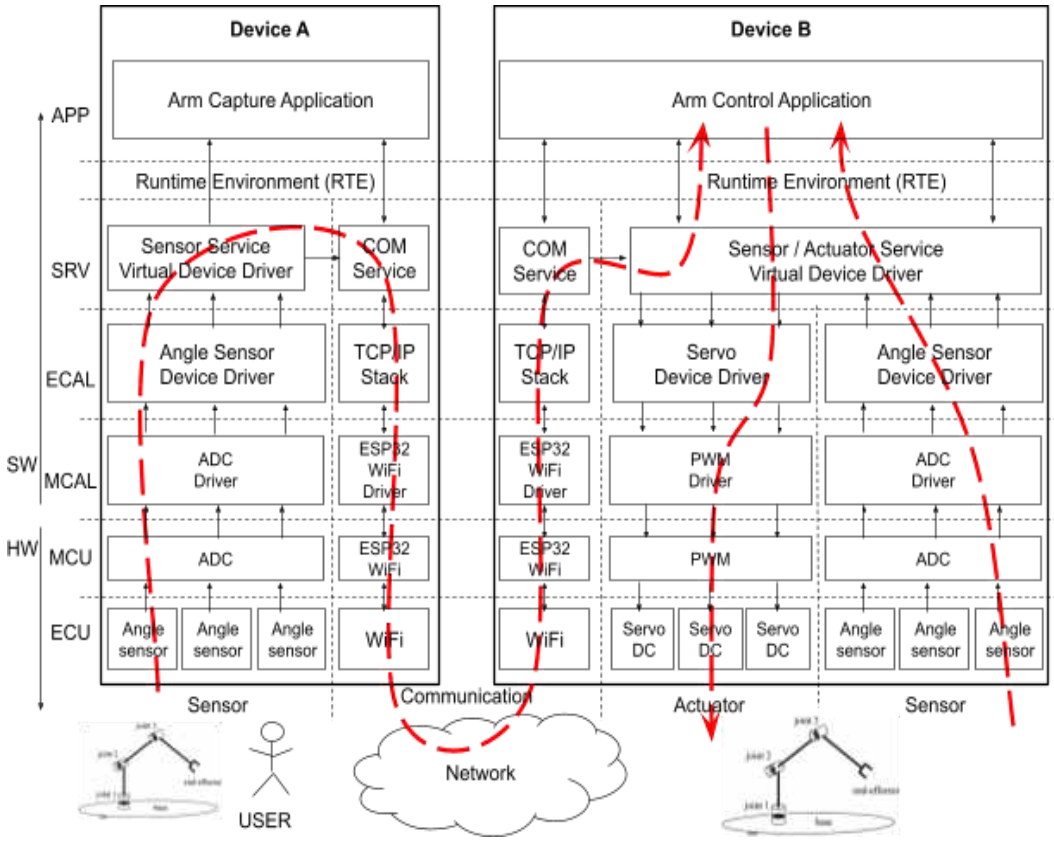


Fig. 3.37. Arhitectura în straturi și fluxul de semnal prin lanțul de comunicare

Sistemul prezentat permite manipularea unui braț robot 3-DOF prin intermediul copiei sale originale sau a unui model software al brațului prin trimiterea prin rețeaua fără fir a unghiurilor de referință articulare pentru brațul actuator sau a coordonatelor carteziene ale capului atunci când este implicată cinematica inversă.

3.3.3 Sistem de control al reflecției luminii

Modelele matematice sunt considerate drept componente reutilizabile dezvoltate pentru scopurile specifice aplicației. În cadrul metodologiei de modelare a sistemelor electronice distribuite, descrise în teză, componentele respective sunt preluate ca subsisteme cu funcții de transfer specifice, care se pot interconecta cu alte componente ale sistemului, conform metodologiei expuse.

Componente ce conțin modelare matematică au fost dezvoltate pentru sistemul de control al reflecției luminii pentru iluminarea ambientală a unei construcții, denumirea de lucru a acestui sistem fiind ”Heliostat”. Dezvoltarea acestui sistem de iluminare interioară prin reflexie a fost realizată pentru a fi instalată pe construcția din pavilionul Republicii Moldova de la expoziția ”Expo Milano 2015”, prezentat în Fig. 3.38.

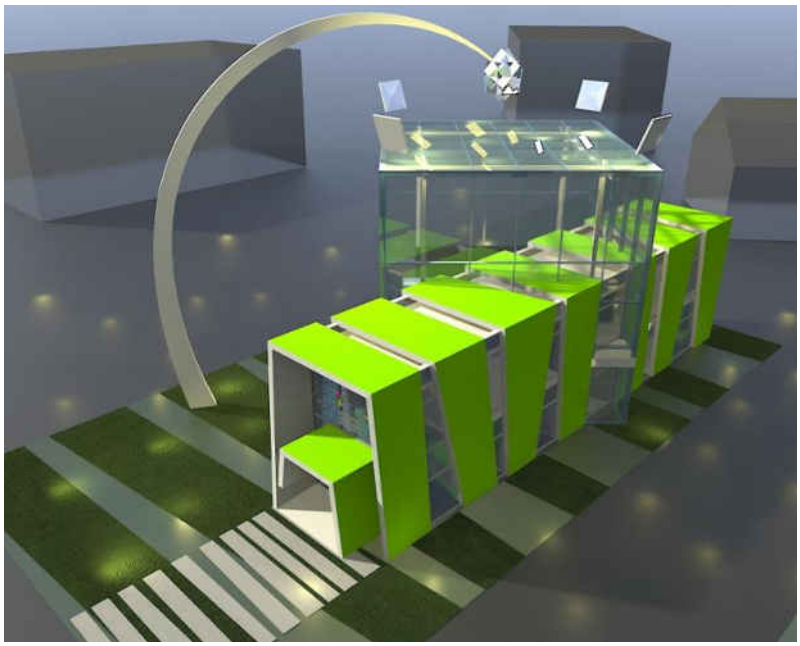


Fig. 3.38. Construcția din pavilionul Republicii Moldova de la expoziția ”Expo Milano 2015”

Modelarea matematică a sistemului de orientare a oglinzilor

Sistemul reprezintă 4 oglinzi situate la colțurile unei clădiri, care au rolul de a urmări mișcarea soarelui și de a reflecta lumina care intră pe o țintă specificată situată în centru, așa cum se arată în Fig. 3.39.

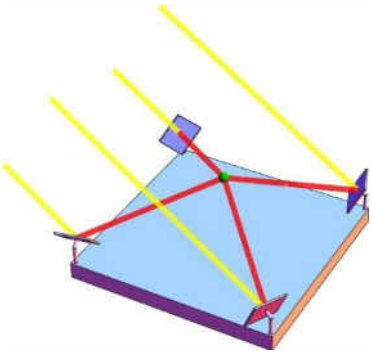


Fig. 3.39. Amplasarea oglinzilor pe colțurile clădirii

Având în vedere poziția soarelui, se calculează vectorul incident pentru raza de intrare $\hat{d}_i$ astfel încât pentru o anumită direcție reflectată (vector reflectat $\hat{d}_s$) sistemul de direcționare a oglinzii își ajustează poziția astfel încât vectorul normal $\hat{n}$ la suprafața oglinzii să satisfacă următoarea ecuație pentru vectorii unitari (3.1):

$$\hat{d}_s = 2\left(\ \hat{n} \cdot \hat{d}_s\right) \cdot \hat{n} - \hat{d}_i \qquad (3.1)$$

Reprezentarea grafică a vectorilor razelor incidente și reflectate este prezentată în Fig. 3.40.

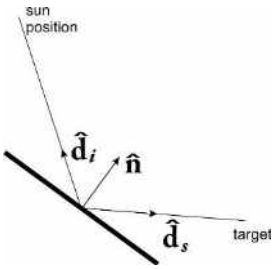


Fig. 3.40. Vectorul incident pentru raza de intrare

unde pentru vectorul normal $\hat{n}$ vom avea (3.2):

$$\hat{n} = \frac{\hat{d}_i + \hat{d}_s}{\left\|\hat{d}_i + \hat{d}_s\right\|} \qquad (3.2)$$

După evaluare se alege soluția astfel încât unghiul dintre $\hat{n}$ și $\hat{d}_i$ să fie mai mic de 90 de grade. Configurația sistemului de orientare este reprezentată în coordonate sferice, Fig. 3.41, luând în considerare direcția nord ca 0 grade (unghi azimut) și măsurând elevația față de sol. Poziția Soarelui este calculată folosind algoritmul PSA. Algoritmul PSA utilizează Universal Time (UT) pentru a elimina incertitudinea cauzată de fusurile orare locale. Locația este introdusă folosind longitudinea și latitudinea, cu minutele și secundele convertite în fracțiuni de grad. Unghiul azimutal este măsurat din nordul adevărat, nu nordul magnetic, iar unghiul zenital este măsurat din verticală. Unghiul de elevație este măsurat de la orizontală.

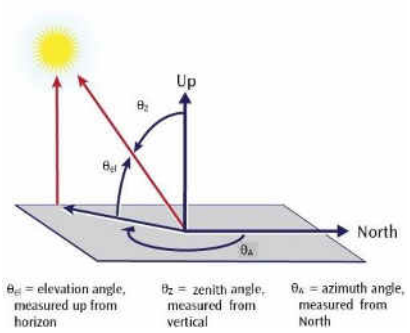


Fig. 3.41. Sistemul de coordonate sferice pentru poziționarea Soarelui

Pentru a menține orientarea luminii către țintă, este necesară o evaluare ciclică a poziției Soarelui, și reevaluarea poziției oglinzii. Diagrama operațională de calcul al vectorului razei de reflecție este prezentată în Fig. 3.42.

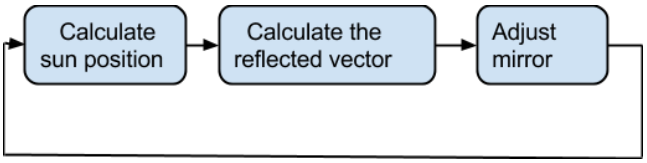


Fig. 3.42. Diagrama operațională de calcul al vectorului razei de reflecție

Conceptual funcționarea sistemului se bazează pe colectarea poziției actuale a oglinzii prin intermediul senzorilor de orientare – accelerometru și magnetometru, evaluarea timpului curent cu un circuit specializat RTC și câteva masuri de protecție, cum ar fi limitarea unghiurilor și protecție împotriva vântului și a altor perturbări care pot cauza funcționarea defectuoasă a oglinzilor. Diagrama conceptuală a sistemului este prezentată în Fig. 3.43

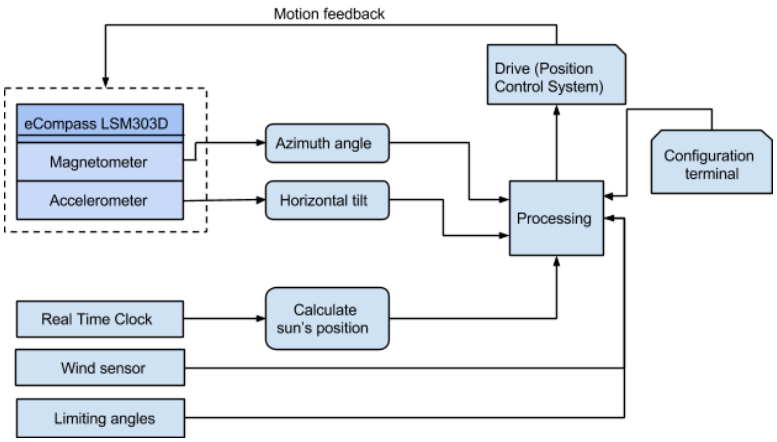


Fig. 3.43. Diagrama conceptuală a sistemului de orientare a oglinzilor

Modelarea arhitecturală a sistemului electronic de direcționării luminii solare

Modelare arhitectură de sistem

Conform cerințelor față de sistemul de orientare a oglinzilor pentru reflecția solară a fost concepută diagrama arhitecturală de interacțiune a componentelor la nivel de sistem, prezentată în

Fig. 3.44. Componenta *Acceleromenter & Giroscope Sensor* servește pentru determinarea poziției curente a oglinzii reflectoare. Componenta *Reflecting Mirror Position Model* implementează modelul matematic de poziționare a oglinzii reflectoare, iar *On/Off Control System* determină deviația poziției curente a vectorului de direcție a oglinzii de la poziția evaluată prin modelul matematic și generează semnale de control pentru componentele de acționare a motoarelor de rotire a oglinzilor pe axele verticală și orizontală prin intermediul componentelor *Vertical Axis Motor* și *Horizontal Axis Motor.*

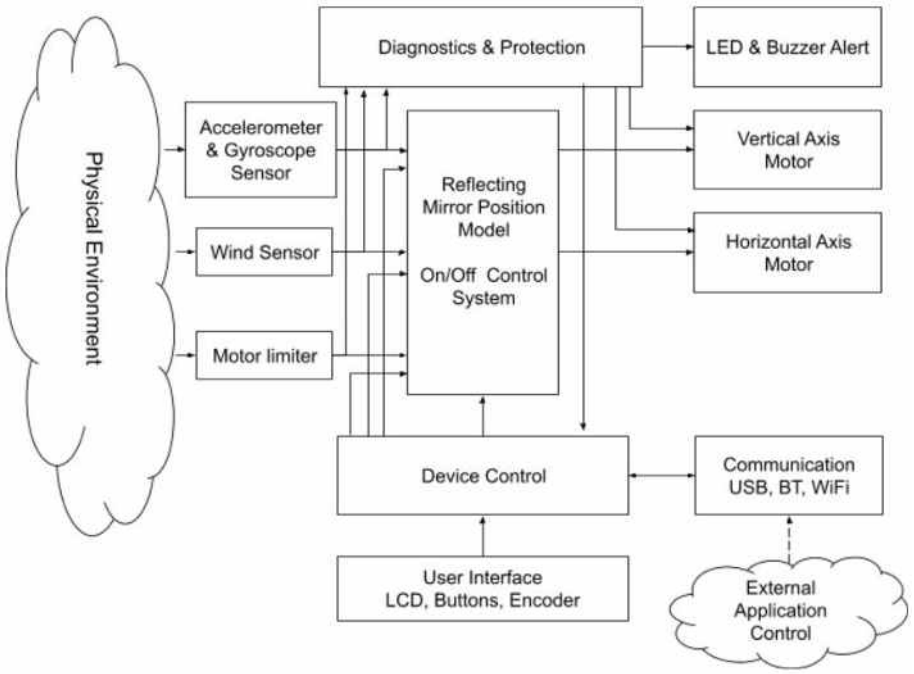


Fig. 3.44. Arhitectura de sistem pentru sistemul de orientare a luminii reflectate

Reieșind din metodologia de modelare a sistemelor propusă în teză, obținem o schiță arhitecturală de sistem pentru componentele la nivel de aplicație, după cum este prezentat în diagrama din Fig. 3.45.

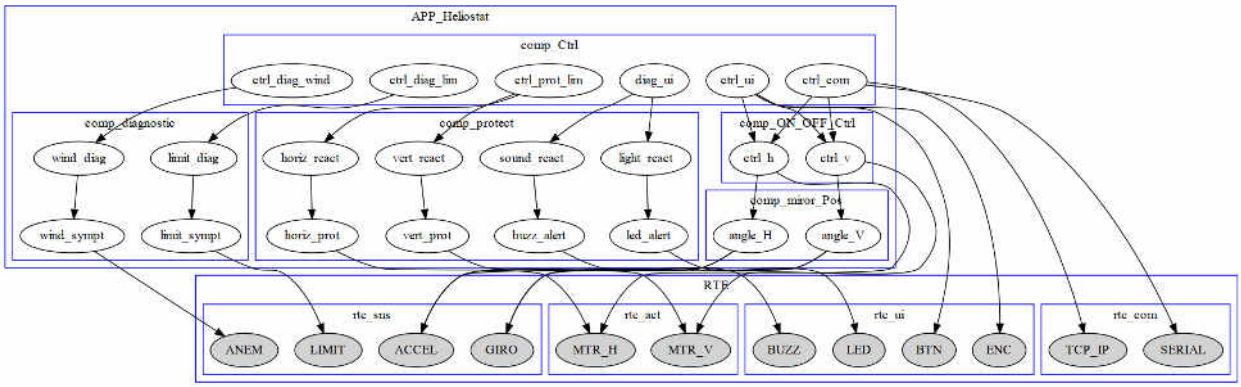


Fig. 3.45. Arhitectura sistemului de orientare generată conform configurației

Sistemul este dotat cu două mecanisme de protecție implementate în cadrul componentei *Diagnostics & Protection,* care activează paralel cu modelul matematic de poziționare. În cazul în care este detectată o situație critică, cum ar fi viteza mare a vântului, această componentă preia controlul motoarelor pentru poziționarea oglinzii în poziție de siguranță maximă, poziție verticală. În cazul în care din diverse motive motoarele conduc oglinda într-o poziție inadmisibilă – mecanismul de protecție preia controlul motoarelor împiedicând continuarea rotirii pe axele respective. Măsurarea vitezei vântului este realizată de componenta *Wind Sensor,* iar detectarea limitelor admisibile de poziționare a oglinzilor este realizată de componenta *Motor Limiter*.

Componenta *Device Control* servește pentru interacțiunea cu sistemul în scop de configurare sau control manual al sistemului fie de la interacțiunea cu utilizatorul printr-un set de butoane și LCD, fie de la distanță prin comunicare cu fir sau fără fir, printr-o aplicație terminal text, fie printr-o aplicație specializată.

Modelare arhitecturală în straturi

Aplicând metodologia descrisă în teză pentru modelarea cu *lanțuri de comunicare*, *arhitectura în straturi* a componentelor de interacțiune pentru componentele generice de senzori, actuatori și comunicare va arăta după cum e prezentată în Fig. 3.46

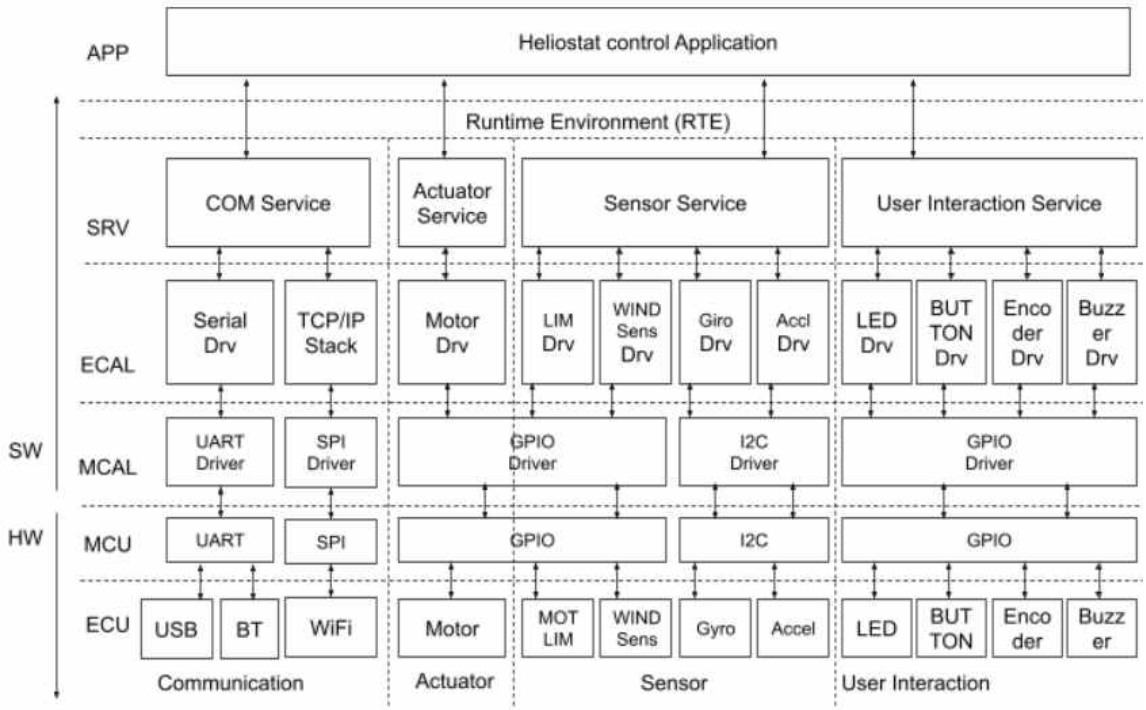


Fig. 3.46. Arhitectura în straturi pentru sistemul de orientare a luminii reflectate

Pentru *arhitectura în straturi* a nivelelor de interacțiune instrumentul de proiectare va genera o schiță arhitecturală, după cum e prezentată în Fig. 3.47.

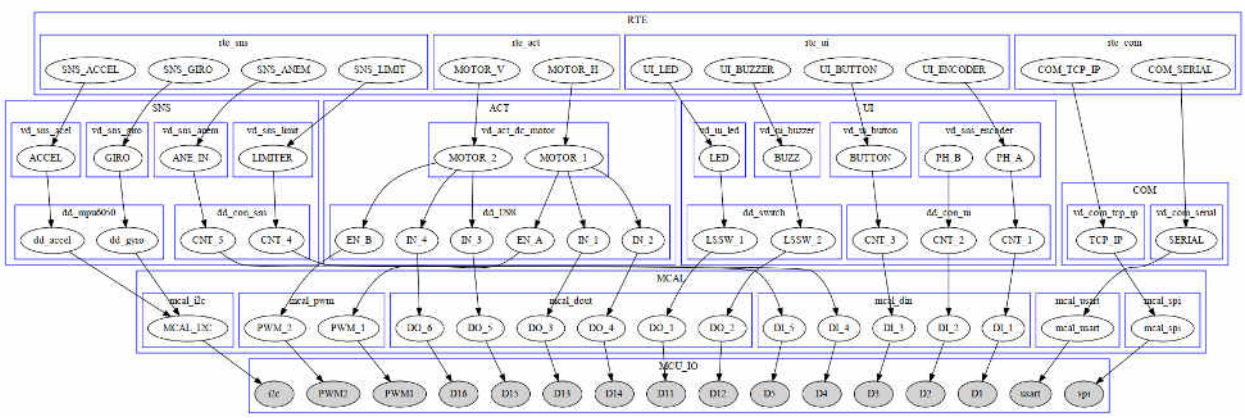


Fig. 3.47. Arhitectura în straturi a sistemului de orientare, generată

3.3.4 Sistem de control al camerei de uscare de fructe

Principiul de elaborare a sistemelor electronice distribuite, expus în teză a fost utilizat pentru elaborarea sistemului de control al camerei de uscare a fructelor. Instalația de uscare prezentată în Fig. 3.48 se întrebuințează pentru uscarea legumelor și fructelor întregi, tăiate rondele sau felii și reprezintă o cameră de 7 m, prin care circulă materialul și agentul de uscare – aerul fierbinte ce îndeplinește funcția de evacuare a umidității din cameră.

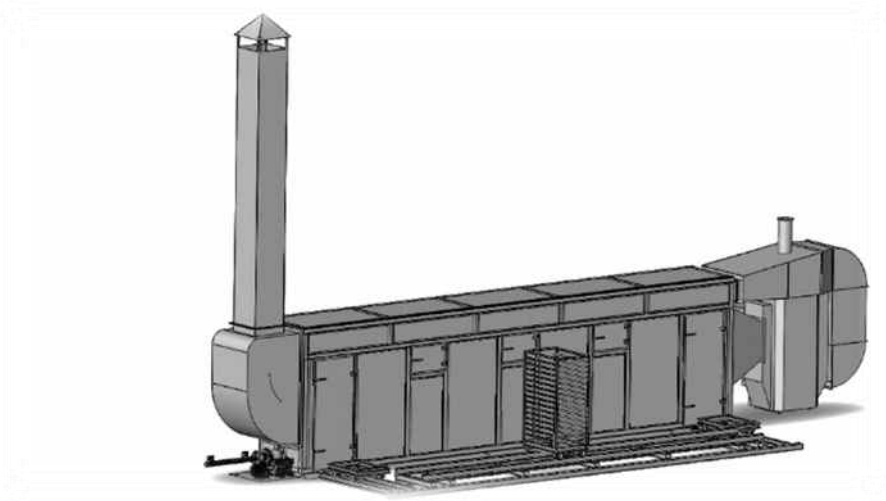


Fig. 3.48. Construcția mecanică a camerei de uscare a fructelor

Ca rezultat al cerințelor de sistem prezentate în detaliu în Anexa 1. și Anexa 9. a rezultat un concept arhitectural al sistemului prezentat în Fig. 3.49.

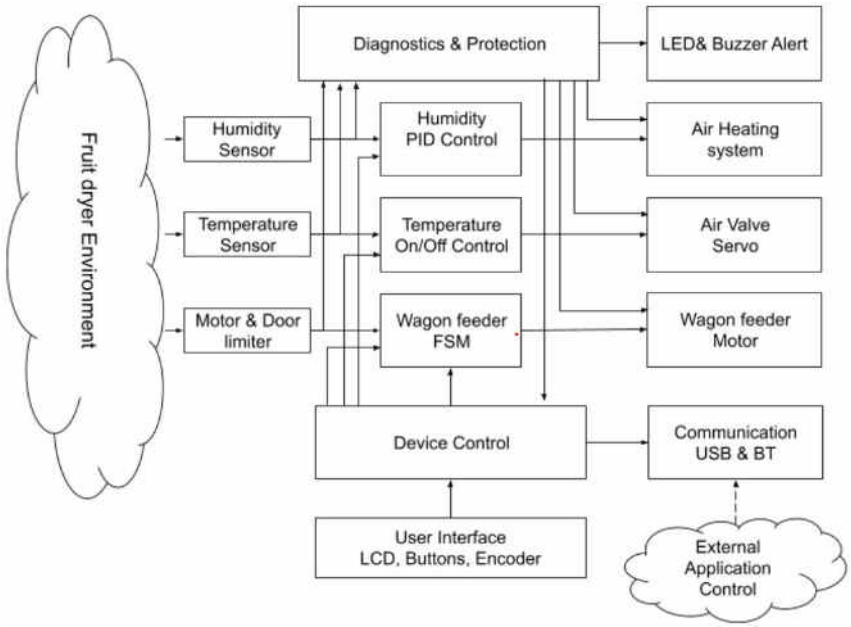


Fig. 3.49. Arhitectura generală a sistemului de control al camerei de uscare

Utilizând instrumentul de modelare prin configurații interconexiuni al componentelor nivel, obținem o schiță arhitecturală a sistemului, după cum urmează în Fig. 3.50.

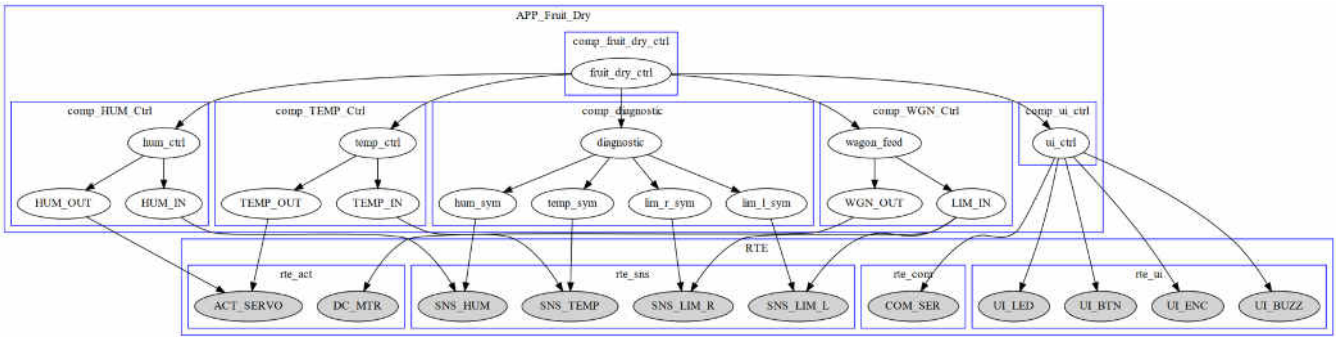


Fig. 3.50. Arhitectura sistemului de control al camerei generat conform configurației

Definirea platformei sistemului bazat pe lanțurile de comunicare ale componentei în straturi.

Aplicând metodologia de modelare a sistemelor electronice distribuite descrisă în teză referitoare la crearea *lanțurilor de comunicare*, și organizând *lanțurile de comunicare* pentru straturile componentei, se obține o arhitectură în straturi, după cum urmează în Fig. 3.51.

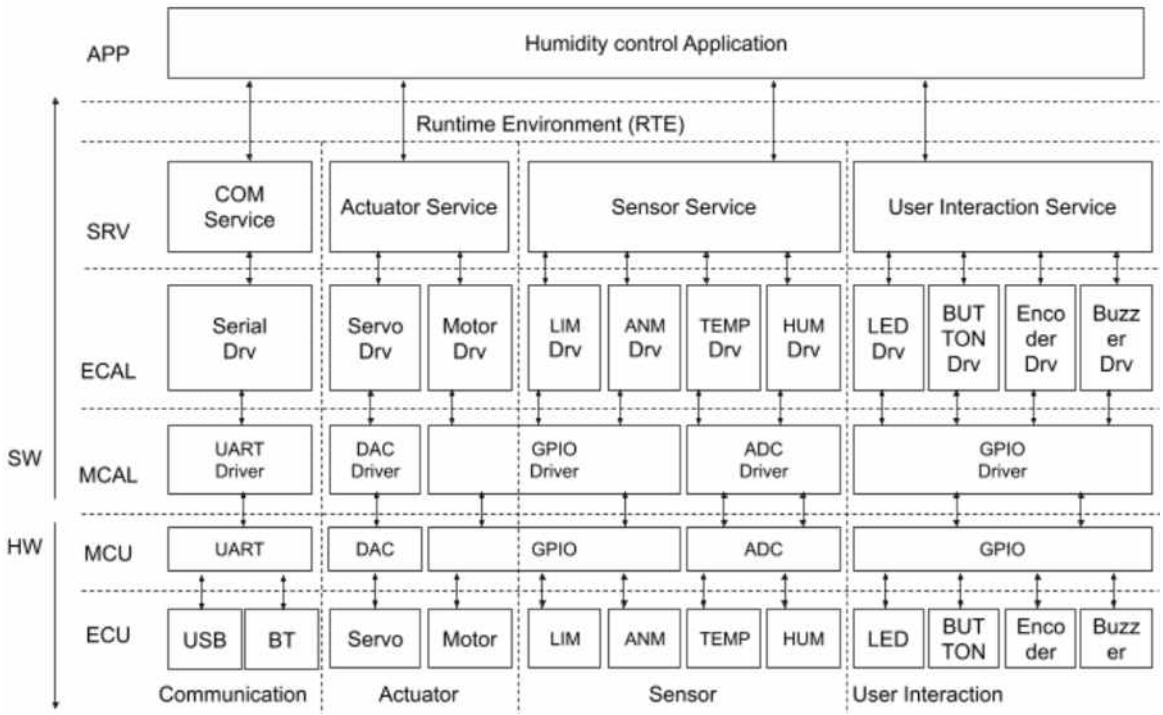


Fig. 3.51. Arhitectura în straturi pentru componentele de interacțiune cu mediul extern

Pentru componentele de platforma de interacțiune cu mediul pentru sistemul de control al camerei de uscare a fructelor, instrumentul de modelare prezentat în teză va genera următoarea schiță *arhitecturală în straturi*, cum e prezentat în Fig. 3.52.

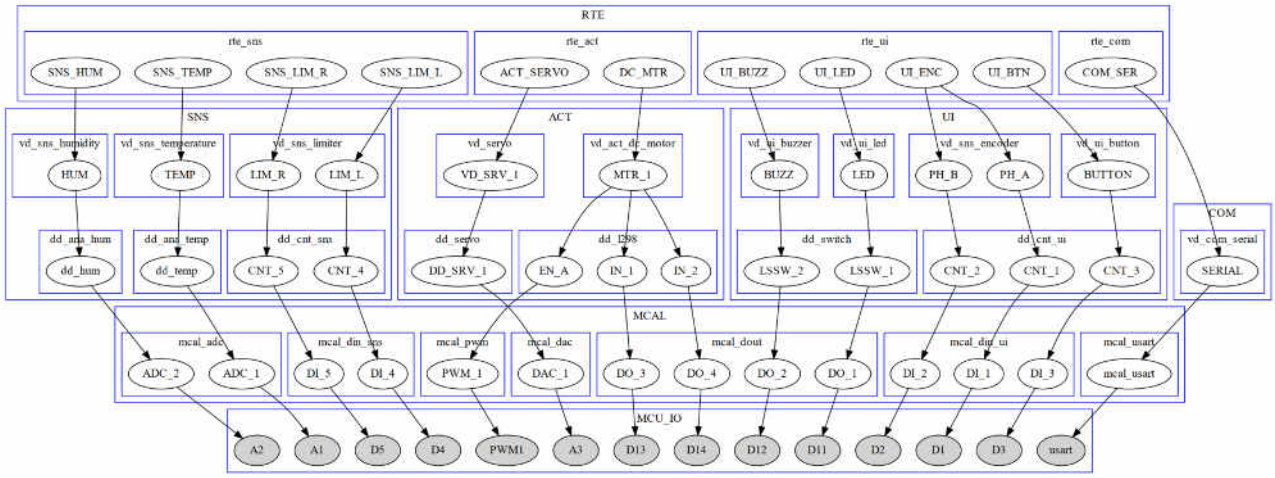


Fig. 3.52. Arhitectura în straturi a sistemului de control al camerei, generată conform configurației

Fiecare dintre componentele menționate în diagrama structurala fie sunt componente reutilizate, după caz, fie definite în cadrul proiectului, și eventual vor fi puse la dispoziție pentru reutilizare pentru următoarele proiecte. Aceste componente înlănțuite pe verticală se pot clasifica ca și componente ce implementează accesul la periferii, cum ar fi senzori, actuatori, interacțiuni

cu utilizatorul sau comunicare. Fiecare dintre *lanțurile de comunicare* ale periferiilor sunt realizate atât în domeniul HW, cât și în SW.

Modelarea componentelor de control al proceselor

Pentru realizarea proceselor de control al instalației de uscare a fructelor au fost implicate mai multe metode și modalități specifice proceselor ce au loc în cadrul procesului tehnologic. Procesele tehnologice pentru care au fost dezvoltate mecanisme de control sunt următoarele:

- Mecanismul de încărcare a camerei de uscare;
- Controlul temperaturii în cameră;
- Controlul umidității în camera de uscare.

Mecanismul de încărcare a camerei de uscare

Pentru controlul mecanismului de încărcare al camerei de uscare este recomandat controlul cu AF, prezentat în capitolul 2. În urma procesului de proiectare al AF al mecanismului de încărcare al camerei de uscare vom obține un comportament, după cum urmează: la apăsarea butonului Start începe mișcarea tijei spre dreapta; Acest lucru continuă până la apăsarea limitatorului de cursă ce indică poziția maximă de dreapta a tijei; apoi începe mișcarea tijei spre stânga până se apasă limitatorul de cursă de indicare a poziției maxime de stânga a tijei. Aceste limitatoare au contactele de tip Normal Închis (Normally Closed – NC) pentru evitarea situațiilor când în lipsa contactului electric de la limitator sau probleme ce țin de cablul defect, tija se deplasează peste poziții extreme, ceea ce duce la deteriorarea părții mecanice și reductorului.

Totodată, algoritmul de acționare a tijei de împingere a stelajelor cu fructe nu începe până nu se sesizează prezența cărucioarelor de la ușa de introducere a stelajelor cu fructe și scoaterea acestora din camera de uscare. Primul cărucior este necesar să fie pe loc, căci, de pe el stelajul cu fructe este împins mai departe în camera de uscare, iar al doilea cărucior trebuie să fie la locul său, fiindcă ultimul cărucior din camera de uscare este împins peste el. În cazul în care acesta nu ar fi la locul stabilit, stelajul cu fructe ar cădea în gol. Limitatoarele de cursă indicatoare a prezenței cărucioarelor au contacte de timpul Normal Deschis (Normally Open - NO), pentru evitarea situațiilor când lipsa conexiunii dintre limitator și blocul electronic de comandă ar indica greșit prezența cărucioarelor.

Motorul tijei de împingere a stelajelor cu fructe este un motor trifazat cu puterea de 1,1 kW, alimentat prin intermediul a două contactoare, pentru rotirea în direcția necesară împingerii stelajelor (spre dreapta) și direcția de retragere în poziția inițială (spre stânga). Inversarea direcției de rotire a motorului se realizează prin inversarea a două faze de alimentare a motorului.

Acționarea contactoarelor se execută prin intermediul Modulului Electronic de Control atașate la pinii digitali de intrare-ieșire.

Controlul temperaturii în camera de uscare

Controlul temperaturii este realizat de către sistemul de control al cazanului care funcționează în baza de control ON-OFF cu histerezis model descris în capitolul 2. Această parte de funcționare a fost integrată în sistem împreună cu cazanul de încălzire a aerului. Temperatura este menținută în mod automatizat în cazan, în camera de uscare – la intrare și ieșire. Diapazonul de temperatură este de la 60°C la 95°C.

Controlul umidității în camera de uscare

Umiditatea aerului este menținută în camera de uscare – la intrare, ieșire, prin intermediul unor clapete dirijate în mod automat, fiind sincronizate între ele, astfel asigurând o umiditate corespunzătoare a produsului finit, independentă de aerul proaspăt care intră parțial în camera de uscare, și cel cu umiditate ridicată, preluată de la produs, care iese parțial din camera de uscare. Acest control al clapetelor este realizat prin modelul PID descris în capitolul 2.

A fost prevăzut că umiditatea produsului supus uscării să nu depășească 10%, în funcție de produsul obținut în urma procesului de uscare. Astfel, în schema de automatizare este prevăzută o dependență între temperatura și umiditatea aerului, ținând cont că temperatura aerului variază independentă de umiditatea lui.

Este prevăzută automatizarea mișcării periodice a vagonetelor în interiorul instalației de uscare. Această deplasare este asigurată de mecanismul de acționare, ce antrenează un melc care efectuează o mișcare liniară periodică în interiorul camerei de uscare, deplasând un vagonet la o distanță egală cu lungimea vagonetului însăși. Punerea în mișcare reversibilă a acestui melc este sincronizată de limitatoare, ce sunt fixate la capetele extreme ale lui. Limitatoarele sunt sincronizate în mod automat cu alte două limitatoare ce sunt fixate în interiorul camerei de uscare, și anume la ușa de intrare a vagonetului, și la ușa de ieșire. Când vagonetul este scos din camera de uscare prin ușa de ieșire, limitatorul de la ieșire imediat dă impuls și melcul deplasează vagonetele la o distanță egală cu lungimea unui vagonet, după care se retrage în poziția inițială. La rândul său muncitorul deplasează un alt vagonet încărcat cu materie primă predestinată uscării prin ușa de intrare a camerei de uscare în spațiul rămas dintre capătul melcului și următoarea vagonetă din uscător. În așa mod ciclul se repetă, asigurând o funcționare continuă a instalației de uscare.

3.4 Concluzii la capitolul 3

1. Au fost identificate etapele procesului de dezvoltare a sistemelor electronice distribuite, în care metodologia de *modelare arhitecturală* cu *lanțuri de comunicare* contribuie considerabil prin automatizarea activităților din cadrul acestor etape.
2. S-a realizat că datorită faptului că sistemele electronice distribuite implică diverse domenii ale ingineriei, cum ar fi Ingineria Mecanică, Ingineria Electrică, ingineria Software, dar și alte domenii interdisciplinare, urmare a unui proces de proiectare bine definit constituie un factor cheie pentru obținerea unui sistem viabil, fazele principale ale acestui proces fiind definirea cerințelor, *modelarea arhitecturală*, implementare pe discipline, testare și validare.
3. Au fost puse bazele de elaborare a unui *produs de program* în scopul automatizării procesului de modelare a sistemelor electronice distribuite prin crearea *lanțurilor de comunicare,* prin intermediul căruia se facilitează aplicarea metodologiei propuse în teză.
4. Au fost definite structurile de metamodele care implementează procesul de *modelare arhitecturală* prin definire de *lanțuri de comunicare* și mecanismele pentru elaborarea produsului program care utilizează proprietățile definite în metamodele, aplică constrângerile de interacțiuni și recomandă selectarea opțiunilor de interconectare prin intermediul interfețelor grafice a programului.
5. A fost justificată utilizarea de *lanțuri de comunicare* pentru realizarea funcțiilor complexe de transfer ale componentelor *arhitecturii în straturi* pentru un sistem electronic distribuit conform conceptului de automatizare propus în teză.
6. S-a constatat că totalitatea de *lanțuri de comunicare* pentru componentele arhitecturii generice propuse cum ar fi senzori, actuatori, interfețe de comunicare, memorie și alimentare cu energie, sunt subiect de re-reutilizare și formează platforma software a sistemului electronic încorporat.
7. S-a constatat că platforma software generică se poate adapta la majoritatea proiectelor cu mici adaptări de configurare automată, luând în considerație că majoritatea sistemelor electronice încorporate tratează în mod similar problemele de interacțiune.
8. S-a demonstrat că metodologia propusă este aplicabilă și la interconectarea componentelor din stratul de aplicație definite conform cerințelor specifice proiectului, care variază considerabil de la un proiect la altul.
9. A fost validată metodologia prin modelarea mai multor proiecte cu aplicarea mecanismelor propuse în teză, unde s-a constatat similaritatea *lanțurilor de comunicare* care definesc platforma software.

10. S-a confirmat faptul că o platformă, odată definită pentru un proiect, poate fi reutilizată și în alte proiecte, cu mici adaptări, efortul major fiind concentrat în stabilirea *lanțurilor de comunicare* ce definesc arhitectura de sistem din stratul de aplicație.
11. A fost dezvoltat un prototip pentru *un sistem de achiziții și construire a hărții mediului* compus dintr-un coordonator și mai multe dispozitive IoT, conform arhitecturii conceptuale propuse pentru un sistem IoT. Acest prototip a validat universalitatea metodei de modelare prin faptul că atât dispozitivele, cât și coordonatorul, sunt implementate urmând același concept.
12. A fost definită o arhitectură pentru o *aplicație robotică de control al brațului 3-DOF*. Soluția permite scalabilitatea și reutilizarea, de exemplu, în cazul în care este necesară o nouă articulație a brațului sau chiar un nou braț robotizat. Respectiv, sistemul poate fi extins prin definirea configurației suplimentare pentru componentele senzorului și actuatorului.
13. A fost validată posibilitatea de integrare a funcționalităților definite prin *modelare matematică,* considerându-le drep t funcții de transfer membre ale *lanțurilor de comunicare* prin modelarea unui *sistem de reflexie a luminii cu oglinzi* pentru iluminarea interiorului unei clădiri.
14. A fost realizat și un proiect pentru controlul *instalației de uscare a fructelor,* urmărindu-se același concept. Prin acest sistem a fost validată universalitatea conceptului ce permite integrarea diverselor subsisteme de control de procese, implicând diverse mecanisme de control, cum ar fi ON-OFF, PID și controlul comportamental cu automate finite.

CONCLUZII GENERALE ȘI RECOMANDĂRI

În urma studiului abordărilor actuale cu privire la sistemele electronice distribuite în vederea identificării aspectelor comune care le caracterizează, s-a identificat că multe dintre abordări au o tendința de generalizare și modularizare. În teză s-au luat ca referință trei tehnologii dintre cele mai avansate în dezvoltarea și aplicabilitatea lor, reprezentând sumar toate nivelele de abstracție: echipament, interacțiune și aplicație. O combinare a acestor trei tehnologii în una generalizată permite dezvoltarea de aplicații cu interacțiuni între noduri-componente pe toate nivelele, cum ar fi: conceptul AUTOSAR la nivel de dispozitiv, cu MQTT globalizare, iar cu ROS aplicații la nivel de sistem. Dezvoltarea unui astfel de concept de *modelare arhitecturală în straturi* cu *lanțuri de comunicare* este unul din obiectivele principale ale acestei teze.

Principalele concluzii:

1. A fost introdus *conceptul de arhitectură generică* a unui dispozitiv electronic încorporat, *ceea ce a condus* la o clasificare a componentelor unui dispozitiv electronic în patru grupuri, în funcție de problema pe care o soluționează – senzori, actuatori, comunicare și interacțiunea cu utilizatorul, fiind completate cu încă două: stocare de date și managementul energiei, indispensabile în funcționarea unui dispozitiv electronic [154].
2. A fost definită o *arhitectură în staturi*, *ceea ce a permis* abstractizarea pe nivele în realizarea componentelor similare pentru toate grupurile de componente unde dispozitivul este asociat cu stratul de servicii, componentele electronice cu stratul de driver, și componenta de interfețe cu microcontrolerul și periferiile [154].
3. A fost introdusă *noțiunea de lanț de comunicare implementat cu funcții de transfer* pentru *achiziția* de informații, *ceea ce a permis* organizarea gestionării și condiționării acesteia de la *senzorul* fizic până la componentele aplicației. *A fost demonstrat* că *lanțul de comunicare* de *acționare* poate fi implementat cu un lanț de funcții de transfer similar cu cel de achiziție, dar invers direcționat [154, 156].
4. *Pentru prima dată* a fost introdus *conceptul de lanț de comunicare comun* pentru organizarea fluxurilor de informație combinând toate tipurile de interacțiuni, cum ar fi *achiziții*, *acționare* și *comunicare*, *ceea ce a condus* la conceptul de *lanț universal de comunicare* pentru dispozitive, reieșind din faptul că în *comunicații* prin rețele la recepție are loc un proces de achiziție semnal *similar* componentelor de *senzor,* iar pentru transmitere procesele similare componentelor de *acționare* completându-se cu tehnologiile de transport date [156, 168].
5. Organizând un *lanț de comunicare* într-un ciclu iterativ, și anume: intrare-ieșire-intrare se poate obține abstracția prin care o aplicație de pe un dispozitiv accesează un senzor atașat la

alt dispozitiv. Această modalitate de abstractizare a interacțiunilor fiind la baza elaborării conceptului de sisteme electronice distribuite *subiect de noutate al tezei – realizarea conceptului de acces la orice de oriunde* [168].

6. A fost constatat că *identificărea simptomelor* și *calificarea de diagnoze* sunt fluxuri de informații derivate asociate achiziției informațiilor, iar *mecanismele de protecție* sunt fluxuri derivate din lanțul de acționare, *fapt care a permis* aplicarea metodologiei descrise în teză pentru proiectarea componentelor specifice pentru asigurarea *siguranței funcționale a sistemelor* [154, 156, 168].
7. A fost folosită noțiunea de *componentă configurabilă* reprezentată de o *colecție de funcții de transfer*, fapt care facilitează *construirea automată* a *lanțurilor de comunicare*, în baza parametrilor specificați în fișiere de definiție în format JSON [170].
8. *A fost implementat* un mecanism de *conectare la servicii de versionare* de resurse software GIT accesibile online pentru o baza de *componente de platformă* oferind posibilități de dezvoltare în grup și schimb de experiență în dezvoltarea pe *resurse reutilizabile*, față de cele dezvoltate în totalitate manual, *acestea aducând* numeroase avantaje - reducerea timpului de realizare, modularitate și altele esențiale [170].
9. *A fost elaborat* un *produs program* în scopul *proiectării sistemelor electronice distribuite*, *ceea ce a permis* elaborarea unui proces de *modelare automată a configurației* sistemului bazată pe metamodele, în format JSON, cât și a generării ulterioare codului sursa pentru dispozitivele electronice, punând la dispoziție automatizări și recomandări pentru selecția de conexiuni de *lanțuri de comunicare* [170].
10. *A fost elaborată o metodologie de modelare a sistemelor electronice distribuite* în scopul asigurării cu instrumente de proiectare, *ceea ce a permis* validarea conceptului *arhitectural în straturi* cu *lanțuri de comunicare* pentru aplicațiile în sisteme din domeniul IoT definit de un set de componente generice [154, 156, 170].
11. Rezultatele cercetărilor din cadrul tezei au fost folosite la *elaborarea de curricule* pentru următoarele cursuri universitare:
    - *Internetul Lucrurilor* - programele de studiu Sisteme Informaționale, Tehnologii Informaționale, nominalizat câștigător, Locul I, în cadrul concursului cursurilor digitale pe platforma educațională ELSE a Universității Tehnice a Moldovei;
    - *Sisteme Încorporate* - programele de studiu Microelectronica și Nano tehnologii, Inginerie Biomedicală, și Inginerie Software;
    - *Aplicații ale Sistemelor Robotice* - programul de studiu Robotică și Mecatronica, în cadrul facultății Calculatoare Informatică și Microelectronică.

12. Metodologia propusă a fost utilizată la elaborarea diverselor *sisteme electronice distribuite*, în special:
    - *Sistem de monitorizare a mediului înconjurător* – colectarea diverșilor parametri fizici ai mediului înconjurător, cum ar fi temperatura, umiditate, luminozitate, CO, mișcare, nivel zgomot prin intermediul instalării distribuite a dispozitivelor electronice dotate cu senzori, cu o construire eventuala harților de mediu în baza datelor colectate din rețeaua sistemului electronic distribuit [155];
    - *Sistem de uscare în bază de pelete pentru fructe și legume* - instalație cu capacitatea de 1,5 tone materie primă pentru compania „Viomarix-Plus" SRL, Rep. Moldova. În rezultatul implementării, a fost instalat un sistem de automatizare la instalația de uscare sus numită, care a contribuit la reducerea consumului de energie cu 15%, constituind 51kW/h.;
    - *Sistem de control al brațelor robotice cu 6 grade de libertate*, care permite interacțiunea între mai multe brațe pentru a executa manipulări mecanice complexe [168];
    - *Sistem de iluminare a interiorului clădirii* cu lumină ambientală prin intermediul a patru oglinzi situate la colțurile unei clădiri, care au rolul de a urmări mișcarea Soarelui și de a reflecta lumina care intră pe o țintă specificată situată în centru. Sistemul a fost proiectat pentru Pavilionul Republicii Moldova la Expoziția de la Milano, 2015, A.O. Asociația Artelor Alternative „ARTWATT", Rep. Moldova;
    - *Sistem de control al unui frigider inteligent* cu funcționalități de control temperatură în camera frigorifică și protecție motor pompă de căldură, dotat cu sistem multimedia pentru monitorizarea a produselor, recomandări de rețete, prototip elaborat pentru compania Micrologic Design Automation SRL, Chișinău, Rep. Moldova;
    - *În scop educațional* a fost elaborată *o placă de dezvoltare,* urmând arhitectura generică a unui sistem încorporat, cu posibilități de experimentare pe parte de senzori, actuatori, interacțiune cu utilizatorul și comunicare, utilizată în procesul de studiu în cadrul lucrărilor de laborator. Echipamente educaționale au fost elaborate și pentru comerț pe piața din Franța, pentru compania Absa-NT. Au fost elaborate două echipamente – "PIC Evaluation Board" [154] similară cu cea de la UTM și "PIC32 Communication Board" dedicat interfețelor de comunicare WiFi, Bluetooth, ZigBee, Ethernet, USB. De asemenea, pentru aceeași companie, a fost proiectată și o platforma robotică mobilă educațională "Road Runner" dotata cu modul GPS și două procesoare, ARM și Raspberry, ce permite experimentare cu sisteme electronice distribuite. Proiectele au fost livrate cu suport software elaborate conform metodologiei propuse în teză.

Direcții de cercetare pentru viitor:

Metodologia propusa urmează a fi dezvoltată în continuare, următorii pași fiind îndreptați spre acoperirea necesităților propuse după cum urmează:

- Dezvoltarea metodologiei pentru generarea de configurații pentru toate dispozitivele unui sistem electronic distribuit dintr-un singur model.
- Elaborarea conceptului de dezvoltare a sistemelor în baza cerințelor – ”Requirement based Development” implicând tehnici de inteligență artificială.
- Elaborarea unui concept de autoorganizare a sistemelor electronice distribuite cu o evoluție dinamică a configurațiilor în timp real.

BIBLIOGRAFIE

1. *The "Only" Coke Machine on the Internet.* Carnegie Mellon University [online]. [citat 07.09.2020]. Disponibil: https://www.cs.cmu.edu/~coke/history_long.txt

2. WEISER, M.; *The Computer for the 21st Century.* UCI Donald Bren School of Information & Computer Sciences [online]. [citat 07.09.2020]. Disponibil: https://www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf ,

3. MATTERN, F., FLOERKEMEIER, C. *From the Internet of Computers to the Internet of Things*. In: Sachs, K., Petrov, I., Guerrero, P. (eds) From Active Data Management to Event-Based Systems and More. Lecture Notes in Computer Science, vol 6462. Springer, Berlin, Heidelberg, 2010. DOI: 10.1007/978-3-642-17226-7_15

4. RAJI R. S., *Smart networks for control*. In: IEEE Spectrum, vol. 31, no. 6, pp. 49-55, June 1994, DOI: 10.1109/6.284793.

5. ZASLAVSKY, Arkady, JAYARAMAN, Prem Prakash. *The Internet of things: Discovery in the Internet of things.* In: Magazine Ubiquity Volume 2015 Issue October, ACM New York, NY, USA. DOI: 10.1145/2822529

6. O'SHAUGHNESSY, Susan, EVETT, Steve, COLAIZZI, Paul, and HOWELL, Terry. *Wireless Sensor Network Effectively Controls Center Pivot Irrigation of Sorghum.* In: Applied Engineering in Agriculture 29, 2013, 853-864 p.. DOI: 10.13031/aea.29.9921.

7. AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., ALEDHARI, M., AYYASH M.. *Internet of things: A survey on enabling technologies,protocols, and applications.* In: IEEE Communications Surveys Tutorials,vol. 17, no. 4, pp. 2347–2376, 2015.

8. FARHAN, L., KHAREL, R., KAIWARTYA, O., Hammoudeh, M., and ADEBISI, B.. *Towards green computing for Internet of Things: Energy orientedpath and message scheduling approach.* In: Sustainable Cities and Society,vol. 38, pp. 195 – 204, 2018.

9. SHANCANG, L., Li, X., Da, and SHANSHAN, Z.. *The Internet of Things: asurvey.* In: Information Systems Frontiers, vol. 17, no. 2, pp. 243–259, Apr 2015.

10. BRICIU, Catalin-Virgil, HEININGER, Ioan Filip Franz, *A new trend in automotive software: AUTOSAR concept.* In: Applied Computational Intelligence and Informatics (SACI), 2013 IEEE 8th International Symposium. DOI: 10.1109/SACI.2013.6608977

11. VERMESAN O., FRIESS P., *Digitising the Industry Internet of Things Connecting the Physical, Digital and Virtual Worlds.* In: ed. River Publishers, Lange Geer 442611 PW Delft The Netherlands, 2016

12. HOFFMAN, F., *Industrial Internet of Things vulnerabilities and threats: What stakeholders need to consider.* In: Issues in Information Systems, Vol 20, Issue 1, pp.119-133, 2019

13. GOLOB, M., AL-ASHAAB, A., *Awareness towards Industry 4.0: key enablers and applications for Internet of Things and Big Data.* In: Proc. Collaborative Networks of Cognitive SystemsChapter: 19th IFIP WG 5.5 Working Conference on Virtual Enterprises, PRO-VE 2018, Cardiff, UK, September 17-19, 2018

14. GOKILAVANI, M., ASHIN, B., UNNIKRISHNAN, K.N., SUNDAR, V., AJAY, V., *A study on IoT architecture for IoT application domains.* In: IJIRIS - International Journal of Innovative Research in Information Security, Volume VI, pp. 43-48. 2019.

15. STAMFORD, Gartner, *Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020*. [online] Conn., December 12, 2013. [citat 07.09.2019]. Disponibil: http://www.gartner.com/newsroom/id/2636073 –

16. *More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020.*ABI Research [online]. [citat 07.09.2019]. Disponibil: https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne/ -

17. *Samsung Intelligent Fridge.* Samsung Official Site [online]. [citat 07.09.2019] Disponibil: http://www.samsung.com/us/explore/family-hub-refrigerator/ -

18. *Technology roadmap: Internet of things.* Wikipedia [online]. [citat 07.09.2019] Disponibil : https://en.wikipedia.org/wiki/Internet_of_things

19. LEE, I., LEE, K.. *The Internet of things (IoT): Applications, investments, and challenges for enterprises.* In: Business Horizons, Elsevier, vol. 58, no. 4, July 2015.

20. GUBBI, J. et al.. *Internet of things (IoT): A vision, architectural elements, and future directions.* In: Future Generation Computer Systems, Elsevier, vol. 29, no. 7, Sept 2013.

21. *IoT Architecture IoT for home automation* https://raw.githubusercontent.com/chheplo/node-sgs/master/artwork/SemanticIoTArchitecture.png - Disponibil: 07.09.2019

22. PEREIRA, C., AGUIAR, A., *Towards efficient mobile M2M communications: survey and open challenges.* Sensors. 2014;14(10):19582–608.

23. MOSKO, M., SOLIS, I., UZUN E., WOOD C., *CCNx 1.0 protocol architecture.* Xerox company, computing science laboratory PARC; 2017.

24. WU, Y., Li, J., STANKOVIC, J., WHITEHOUSE, K., SON, S., and KAPITANOVA, K., *Run time assurance of application-level requirements in wireless sensor networks.* In: Proc. 9th ACM/IEEE international conference on information processing in sensor networks, Stockholm, Sweden, 21–16 April 2010. p. 197–208.

25. KUMAR, S., TIWARI, P. and ZYMBLER, M. *Internet of Things is a revolutionary approach for future technology enhancement: a review*. In: J Big Data 6, 111 (2019). DOI: 10.1186/s40537-019-0268-2

26. FARHAN L., KHAREL R., KAIWARTYA, O., QUIROZ-CASTELLANOS M., ALISSA A. and ABDULSALAM, M., *A Concise Review on Internet of Things (IoT) -Problems, Challenges and Opportunities*. In: 2018 11th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), 2018, pp. 1-6, doi: 10.1109/CSNDSP.2018.8471762.

27. SETHI, Pallavi, SARANGI, Smruti R., *Internet of Things: Architectures, Protocols, and Applications.* In: Journal of Electrical and Computer Engineering, vol. 2017, Article ID 9324035, 25 pages, 2017. DOI: 10.1155/2017/9324035

28. HAKKIGBERG, C.. *Experimental Evaluation of LoRaWAN in Indoor and Outdoor Environment.* In: Mater Thesis, University of Twente, Faculty of Electrical Engineering, Mathematics, and Computer Science, 2016.

29. LAVRIC, Alexandru. *LoraWAN communication protocol: The new era of IoT.* In: IEEE International Conference on Development and Application Systems (DAS), June 2018.

30. DINA, Ibrahim, DINA, Hussein, *Internet of Things Technology based on LoRaWAN Revolution.* In: 2019 10th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2019, pp. 234-237, doi: 10.1109/IACS.2019.8809176.

31. LAVRI,C A., POPA, V.. *Internet of things and lora; low-power widearea networks: A survey.* In: 2017 International Symposium on Signals, Circuits and Systems (ISSCS), July 2017, pp. 1–5.

32. CHEONG, S. P., BERGS, J., HAWINKEL, C., and FAMAEY, J.. *Comparison of LoRaWAN classes and their power consumption.* In: Proc. of the 2017 IEEE Symposium on

Communications and Vehicular Technology (SCVT), Leuven, Belgium, 14 November 2017; pp. 1–6.

33. WIXTED, A. J., KINNAIRD, P., LARIJANI, H., TAIT, A., AHMADINIA, A. A., STRACHAN, N.. *Evaluation of LoRa and LoRaWAN for wireless sensor networks.* In: Proc. of the 2016 IEEE SENSORS, Orlando, FL, USA, 30 October–3 November 2016; pp. 1–3.

34. NEUMANN, P., MONTAVONT, J., and NOOL, T., *''Indoor deployment of lowpower wide area networks (LPWAN): A LoRaWAN case study,"* In: Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMOB), 2016, pp. 1–8.

35. AUGUSTIN, A., Yi J., CLAUSEN, T., and TOWNSLEY, W.. *A Study of LoRa: Long Range and Low Power Networks for the Internet of Things.* In: Sensors, vol. 16, no. 12, pp. 1466-1484, Sep. 2016.

36. MIGABO, E., DJOUANI, K., KURIEN, A., and OLWAL, T.. *A Comparative Survey Study on LPWA Networks: LoRa and NB-IoT.* In: Future Technologies Conference (FTC), Canada, Nov.2017, pp. 1045-1051.

37. KHUTSOANE, O., ISONG B., and ABU-MAHFOUZ, A. M., *IoT devices and applications based on LoRa/LoRaWAN.* In: Proc. Annu. Conf. IEEE Ind. Electron. Soc., Beijing, China, Oct./Nov. 2017, pp. 6107–6112.

38. VANGELISTA, L., ZANELLA, A., ZORZI, M., *Long-range IoT technologies: the dawn of LoRa*. In: Proc. EAI Int. Conf. Future Access Enablers for Ubiquitous and Intelligent Infrastructures (Fabulous), Sep. 2015, pp. 51-58.

39. JAIN, R., *Low Power WAN protocols for IoT: IEEE 802.11ah and LoRaWAN.* Lectures, Washington University, Saint Louis, 2016.

40. WIXTED, A. J., KINNAIRD, P., LARIJANI, H., TAIT, A., AHMADINIA, A., and STRACHAN, N.. *Evaluation of LoRa and LoRaWAN for wireless sensor networks.* In: IEEE SENSORS, 2016, pp. 1–3

41. KHUTSOANE, O., ISONG, B. and ABU-MAHFOUZ, A. M., *IoT devices and applications based on LoRa/LoRaWAN.* In: IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, 2017, pp. 6107-6112, doi: 10.1109/IECON.2017.8217061.

42. AGRAWAL, P. and BHURARIA, S., *Near field communication.* In: SETLabs Bridfings, vol. 10, no. 1, pp. 67–74, 2012.

43. COSKUN, V., OZDENIZCI, B., and OK, K., *A survey on near field communication (NFC) technology.* In: Wireless Personal Communications, vol. 71, no. 3, pp. 2259–2294, 2013.

44. CURRAN, K., MILLAR A., and MC GARVEY, C.. *Near Field Communication.* In: International Journal of Electrical and Computer Engineering (IJECE), vol. 2, no. 3, 2012.

45. ARORA, V. K., SHARMA, V., and SACHDEVA, M., *On QoS evaluation for ZigBee incorporated Wireless Sensor Network (IEEE 802.15. 4) using mobile sensor nodes.* In: Journal of King Saud University-Computer and Information Sciences, 2018.

46. VU, V. T., QUYEN, T. V., TRUONG, L. H., LE, A. M., NGUYEN, C. V., and NGUYEN, M. T., *Energy efficient approaches in wireless sensor networks.* In: ICSES Transactions on Computer Networks and Communications, 6(1), 2020, 1-10p.

47. NGUYEN M., NGUYEN, H., MASARACCHIA, A., and NGUYEN, C., *Stochastic-based power consumption analysis for data transmission in wireless sensor networks.* In: EAI Endorsed Transactions on Industrial Networks and Intelligent Systems, 6(19), 2019

48. MARTINEZ-SANDOVAL, R., GARCIA-SANCHEZ, A. J., GARCIA-SANCHEZ, F., GARCIA-HARO J., and FLYNN, D.. *A comprehensive WSN-based approach to efficiently manage a smart grid.* In: Sensors, 14(10) , 2014, 18748-18783.

49. DENG, X., He, T., He, L., Gui, J., and Peng, Q.. *Performance analysis for IEEE 802.11 s wireless mesh network in smart grid.* In: Wireless Personal Communications, 96(1), 2017. 1537-1555.

50. ANNOR-ASANTE, M., and PRANGGONO, B.. *Development of smart grid testbed with low-cost hardware and software for cybersecurity research and education*. In: Wireless Personal Communications, 101(3), 2018. 1357-1377.

51. ZHANG, Y., ZHANG, S., and DING, Y.. *Research on the influence of sensor network communication in the electromagnetic environment of smart grid.* In: Journal of Electrical and Computer Engineering, 2016.

52. RAMESH, M. V., PRABHA, R., THIRUGNANAM, H., DEVIDAS, A. R., Raj, D., Anand, S., and Pathinarupothi, R. K. *Achieving sustainability through smart city applications: protocols, systems and solutions using IoT and wireless sensor network.* In: CSI Transactions on ICT, 8, 2020. 213-230 p.

53. ZHAN, M., WU, J., WEN, H., and ZHANG, P.. *A novel error correction mechanism for energy-efficient cyber-physical systems in smart building.* In: IEEE Access, 6, (2018). 39037-39045.

54. DING, F., SONG, A., ZHANG, D., TONG, E., PAN, Z. and YOU, X.. *Interference-Aware Wireless Networks for Home Monitoring and Performance Evaluation.* In: IEEE Transactions on Automation Science and Engineering, vol. 15, no. 3, July 2018, pp. 1286-1297, DOI: 10.1109/TASE.2017.2778303.

55. PENG, C., and QIAN, K.. *Development and application of a ZigBee-based building energy monitoring and control system*. In: The Scientific World Journal, 2014.

56. HUANG, Li-Chien, et al. *A ZigBee-based monitoring and protection system for building electrical safety.* In: Energy and Buildings 43.6 (2011): 1418-1426.

57. GHARGHAN, Sadik Kamel, ROSDIADEE Nordin, and MAHAMOD Ismail. *A wireless sensor network with soft computing localization techniques for track cycling applications.* In: Sensors 16.8 (2016): 1043.

58. GHARGHAN, Sadik K., ROSDIADEE Nordin, and MAHAMOD Ismail. *An ultra-low power wireless sensor network for bicycle torque performance measurements.* In: Sensors 15.5 (2015): 11741-11768.

59. GHARGHAN, Sadik Kamel, ROSDIADEE, Nordin, and MAHAMOD, Ismail. *Statistical validation of performance of ZigBee-based wireless sensor network for track cycling*. In: International Conference on Smart Sensors and Application (ICSSA). IEEE, 2015.

60. GHARGHAN, Sadik K., et al. *Accurate wireless sensor localization technique based on hybrid PSO-ANN algorithm for indoor and outdoor track cycling*. In: IEEE Sensors Journal 16.2 (2015): 529-541.

61. MAHAPATRA, Ranjan Kumar, and N. S. V. Shet. *Localization based on RSSI exploiting gaussian and averaging filter in wireless sensor network*. In: Arabian Journal for Science and Engineering 43.8 (2018): 4145-4159.

62. BLANCO-NOVOA, Óscar, et al. *An electricity price-aware open-source smart socket for the Internet of energy*. In: Sensors 17.3 (2017): 643.

63. KATYARA, Sunny, et al. *Monitoring, control and energy management of smart grid system via WSN technology through SCADA applications*. In: Wireless Personal Communications 106.4 (2019): 1951-1968.

64. CHINCOLI, Michele, et al. *Power control in wireless sensor networks with variable interference*. In: Mobile Information Systems 2016 (2016).

65. GILL, S. P. S., SURYADEVARA N. K., and MUKHOPADHYAY S. C.. *Smart Power monitoring system using wireless sensor networks*. In: 2012 Sixth International Conference on Sensing Technology (ICST). IEEE, 2012.

66. VALENZUELA, Alejandro A. *Multisensor system for energy consumption awareness in large buildings*. In: 2012 International Conference on Smart Grid Technology, Economics and Policies (SG-TEP). IEEE, 2012.

67. BATISTA, N. C., et al. *Photovoltaic and wind energy systems monitoring and building/home energy management using ZigBee devices within a smart grid*. In: Energy 49 (2013): 306-315.

68. ALOBAIDY, Haider A.H., HIKMAT, N. Abdullah, and TARIQ, M. Salman. *Implementation and performance evaluation of WSN for energy monitoring application*. In: Eng. and Tech. Journal 33.7 (2015): 1555-1568.

69. SURYADEVARA, Nagender Kumar, et al. *WSN-based smart sensors and actuator for power management in intelligent buildings*. In: IEEE/ASME transactions on mechatronics 20.2 (2014): 564-571.

70. SUBRAHMANYAM, V., et al. *A low power minimal error IEEE 802.15. 4 Transceiver for heart monitoring in IoT applications*. In: Wireless Personal Communications 100.2 (2018): 611-629.

71. KONE, Cheick Tidjane, ABDELHAKIM, Hafid, and MUSTAPHA, Boushaba. *Performance management of IEEE 802.15. 4 wireless sensor network for precision agriculture*. In: IEEE Sensors Journal 15.10 (2015): 5734-5747.

72. VARGHESE, Susan G., et al. *Comparative study of ZigBee topologies for IoT-based lighting automation*. In: IET Wireless Sensor Systems 9.4 (2019): 201-207.

73. NGUYEN, Minh, et al. *Wireless Communication Technologies and Applications for Wireless Sensor Networks: A Survey*. In: ICSES Transactions on Computer Networks and Communications 5.1 (2019): 1-15.

74. MINH, Nguyen T., et al. *Electromagnetic field based WPT technologies for UAVs: A comprehensive survey*. In: Electronics 9.3 (2020): 461.

75. LE, A. M., TRUONG, L. H., QUYEN, T. V., NGUYEN, C. V., NGUYEN, M. T. *Wireless Power Transfer Near-field Technologies for Unmanned Aerial Vehicles (UAVs): A Review. EAI Endorsed Trans*. In: Ind. Netw, Intell. Syst 7 (2020): 1-18.

76. NGUYEN, C. V., QUYEN, T. V., LE A. M., TRUONG, L. H., NGUYEN, M. T. *Advanced Hybrid Energy Harvesting Systems for Unmanned Aerial Vehicles (UAVs)*. In: Adv. Sci. Technol. Eng. Syst. J 5 (2020): 34-39.

77. QUYEN, T. V., NGUYEN, C. V., LE, A. M., and NGUYEN, M. T.. *Optimizing hybrid energy harvesting mechanisms for UAVs.* In: EAI Endorsed Transactions on Energy Web, 7(30), e8. (2020).

78. MOHAMMED, Alsultan, , OZTOPRAK, Kasim, and HASSANPOUR, Reza. *Power aware routing protocols in wireless sensor network*. In: IEICE Transactions on Communications 99.7 (2016): 1481-1491.

79. NGUYEN, M. T., and TEAGUE, K. A. *Compressive sensing based random walk routing in wireless sensor networks*. In: Ad Hoc Networks, 54, 99-110. (2017).

80. NGUYEN, Minh Tuan. *Minimizing energy consumption in random walk routing for wireless sensor networks utilizing compressed sensing*. In: 2013 8th International Conference on System of Systems Engineering. IEEE, 2013.

81. COBOI, Alberto , NGUYEN, Van-Cuong and NGUYEN, Minh , DUY, Nguyen and TRAN, Thang. *An Analysis of ZigBee Technologies for Data Routing in Wireless Sensor Networks.* In: ICSES Trans. Comput. Networks Commun. (ITCNC), vol. X, no. August, pp. 1–10, 2021

82. LIU, H., BOLIC, M., NAYAK, A., and STOJMENOVIĆ, I.. *Taxonomy and challenges of the integration of RFID and wireless sensor networks*. In: IEEE Network, vol. 22, no. 6, pp. 26–32, 2008.

83. MITROKOTSA, A. and DOULIGERIS, C.. *Integrated RFID and sensor networks: architectures and applications*. In: RFID and Sensor Networks: Architectures, Protocols, Security and Integrations, pp. 511–535, Auerbach Publications, 2009.

84. SWAMI, Ananthram, et al., eds. *Wireless sensor networks: signal processing and communications perspectives.* In: John Wiley & Sons, 2007.

85. AKYILDIZ, Ian F., and MEHMET Can Vuran. *Wireless sensor networks.* In: Vol. 4. John Wiley & Sons, 2010.

86. AJU, Omojokun G. *A survey of zigbee wireless sensor network technology: Topology, applications and challenges*. In: International Journal of Computer Applications 130.9 (2015): 47-55.

87. *What is Zigbee Technology? Architecture, Topologies and Applications.* Electronics HUB [online]. [citat 07.09.2020]. Disponibil: http://www.electronicshub.org/zigbee-technology –

88. OBAID, Thoraya, et al. *ZigBee Technology and its application in Wireless Home Automation systems: a survey*. In: International Journal of Computer Networks & Communications 6.4 (2014): 115.

89. CENGIZ, Gezer, and BURATTI, Chiara. *A ZigBee smart energy implementation for energy efficient buildings*. In: 2011 IEEE 73rd Vehicular Technology Conference (VTC Spring). IEEE, 2011.

90. CLARKE, Malcolm, et al. *Interoperable end-to-end remote patient monitoring platform based on IEEE 11073 PHD and ZigBee health care profile*. In: IEEE Transactions on Biomedical Engineering 65.5 (2017): 1014-1025.

91. ZHENG, Li. *ZigBee wireless sensor network in industrial applications*. In: 2006 SICE-ICASE International Joint Conference. IEEE, 2006.

92. AJU, Omojokun G. *A survey of zigbee wireless sensor network technology: Topology, applications and challenges*. In: International Journal of Computer Applications 130.9 (2015): 47-55.

93. ESPINOSA-FALLER, Francisco J., and RENDÓN-RODRÍGUEZ Guillermo E.. *A ZigBee wireless sensor network for monitoring an aquaculture recirculating system*. In: Journal of applied research and technology 10.3 (2012): 380-387.

94. CHAITANYA, N. Krishna, KUMAR, G. Anand, and KUMARI, P. Aruna. *ZigBee based wireless sensing platform for monitoring agriculture environment*. In: International Journal of computer applications 83.11 (2013).

95. DINESHKUMAR, Jaiswar, , and SANJNA, S. Repal. *Wireless mesh communication using ZigBee technology for military applications*. In: Int. J. Sci. Res 5.5 (2016).

96. BATISTA, N. C., et al. *Photovoltaic and wind energy systems monitoring and building/home energy management using ZigBee devices within a smart grid*. In: Energy 49 (2013): 306-315.

97. BOSCH, R.. *CAN Specification 2.0*. In: Postfach, Stuttgart, Germany: Robert Bosch GmbH, 1991.

98. SHORT, M. and PONT, M.J.. *Fault-Tolerant Time-Triggered Communication Using CAN*. In: IEEE Transactions on Industrial Informatics, vol. 3, no. 2, pp. 131-142, May 2007, DOI: 10.1109/TII.2007.898477.

99. BARRANCO, M., PROENZA, J., and ALMEIDA, L.. *Boosting the Robustness of Controller Area Networks: CANcentrate and ReCANcentrate*. In: IEEE Computer, Vol. 42, No. 5, pp. 66–73.

100. KIM, S., LEE E., CHOI, M., JEONG, H. and SEO, S.. *Design Optimization of Vehicle Control Networks. IEEE Transactions on Vehicular Technology,* Vol. 60, No. 7, pp. 3002-3016, 2011.

101. COOK, J. A., KOLMANOVSKY, I. V., MCNAMARA, D., E. NELSON, C. and PRASAD, K. V.. Control, computing and communications: *Technologies for the twenty-first century model T.* In: Proceedings of the IEEE—Special Issue on Automotive Power Electronics and Motor Drives, Vol. 95, No. 2, pp. 334–355, 2007.

102. SHORT, Michael, SHEIKH, Imran, and RIZVI, Syed.. *A Transmission Window Technique for CAN Networks.* In: Journal of Systems Architecture. 69. 15-28. (2016) DOI: 10.1016/j.sysarc.2016.07.001.

103. *FlexRay Automotive Communication Bus Overview*. Nationa Instrument Official Site [online]. [citat 20 09 2019] Disponibil: https://www.ni.com/en-us/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/flexray-automotive-communication-bus-overview.html

104. KOPETZ, H.. *A Comparison of CAN and TTP.* Annual Reviews in Control, Vol. 24, pp. 177–188, 2000.

105. AYAVOO D., PONT M.J., SHORT M., and PARKER S.. *Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems*. In: Microprocessors and Microsystems, Vol. 31, No. 5, pp. 326-334, 2007.

106. LEEN, G. and HEFFERNAN, D.. *TT-CAN: a new time-triggered controller area network*. In: Microprocessors and Microsystems, Vol. 26, No. 2, pp. 77-94, 2002.

107. MIUCIC, R., MAHMUD, S.M. and POPOVIC, Z.. *An Enhanced Data Reduction Algorithm for Event Triggered Networks.* In: IEEE Transactions on Vehicular Technology, Vol. 58, No. 6, pp. 2663-2678, 2009

108. MASHAL, I., ALSARYRAH, O., CHUNG, T.Y., YANG C.Z., KUO, W.H., and AGRAWAL, D. P.. *Choices for interaction with things on Internet and underlying issues*. In: Ad Hoc Networks, vol. 28, pp. 68–90, 2015.

109. SAID, O. and MASUD, M.. *Towards Internet of things: survey and future vision*. In: International Journal of Computer Networks, vol. 5, no. 1, pp. 1–17, 2013.

110. WU, M., LU, T.J., LING, F.Y., SUN, J., and DU, H.Y.. *Research on the architecture of Internet of things*. In: Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE '10), vol. 5, pp. V5-484–V5-487, IEEE, Chengdu, China, August 2010.

111. KHAN, R., KHAN, S. U., ZAHEER, R., and KHAN, S.. *Future Internet: the Internet of things architecture, possible applications and key challenges*. In: Proceedings of the 10th International Conference on Frontiers of Information Technology (FIT '12), pp. 257–260, December 2012.

112. NING, H. and WANG, Z.. *Future Internet of things architecture: like mankind neural system or social organization framework?*. In: IEEE Communications Letters, vol. 15, no. 4, pp. 461–463, 2011.

113. *What is middleware?*. Red Hat Official Site [online]. [citat 21.08.2021]. Disponibil: https://www.redhat.com/en/topics/middleware/what-is-middleware -

114. BANDYOPADHYAY, S., SENGUPTA, M., MAITI, S., and DUTTA, S.. *Role of middleware for Internet of things: a study*. In: International Journal of Computer Science & Engineering Survey, vol. 2, no. 3, pp. 94–105, 2011.

115. CHAQFEH, M. A. and MOHAMED, N.. *Challenges in middleware solutions for the Internet of things*. In Proceedings of the 13th International Conference on Collaboration Technologies and Systems (CTS '12), pp. 21–26, Denver, Colo, USA, May 2012.

116. RAZZAQUE, M. A., MILOJEVIC-JEVRIC, M., PALADE A., and CLA S.. *Middleware for Internet of things: a survey*. In: IEEE Internet of Things Journal, vol. 3, no. 1, pp. 70–95, 2016.

117. SOLDATOS, J., KEFALAKIS, N., HAUSWIRTH, M. et al.. *Openiot: open source Internet of-things in the cloud*. In: Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia, September 18, 2014. Invited Papers, vol. 9001 of Lecture Notes in Computer Science, pp. 13–25, Springer, Berlin, Germany, 2015.

118. RANGANATHAN, A., AL-MUHTADI, J., CHETAN, S., CAMPBELL, R., and MICKUNAS, M. D.. *Middlewhere: a middleware for location awareness in ubiquitous computing applications*. In: ACM/IFIP/USENIX International Conference on Distributed

Systems Platforms and Open Distributed Processing Middleware 2004, pp. 397–416, Springer, New York, NY, USA, 2004.

119. EISENHAUER, M., ROSENGREN, P., and ANTOLIN P.. *A development platform for integrating wireless devices and sensors into ambient intelligence systems*. In: Proceedings of the 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops (SECON Workshops '09), pp. 1–3, IEEE, Rome, Italy, June 2009.

120. SONG, Z., A. CÁRDENAS, A., and MASUOKA, R.. *Semantic middleware for the Internet of things*. In: Proceedings of the 2nd International Internet of Things Conference (IoT '10), December 2010.

121. KATASONOV, A., KAYKOVA, O., KHRIYENKO, O., NIKITIN, S., and TERZIYAN, V.. *Smart semantic middleware for the Internet of things*. In: Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics (ICINCO '08), pp. 169–178, Funchal, Portugal, May 2008.

122. MACLEAN, Fiona, JONES, Derek, CARIN-LEVY, Gail and HUNTER, Heather. *Understanding Twitter.* In: British Journal of Occupational Therapy. 76(6):295. (2013). DOI: 10.4276/030802213X13706169933021.

123. ATMOKO, Rachmad, RIANTINI, R and HASIN, M. *IoT real time data acquisition using MQTT protocol.* In: Journal of Physics: Conference Series. 853. 012003. (2017). DOI: 10.1088/1742-6596/853/1/012003.

124. SHIVA, Shankar J, PALANIVEL, S., VENKATESWARLU, S. China, SOWMYA, M.. *MQTT in Internet of Things.* In: International Research Journal of Engineering and Technology(IRJET) pp796-798. (2019).

125. *MQTT broker - a Twitter for machines.* https://medium.com/@noobino/mqtt-broker-a-twitter-for-machines-b624ea2773b1 Disponibil: Jun 20, 2020.

126. ANDERSON, Mike, *Introduction to the Robot Operating System (ROS) Middleware*. In: Embedded Linux Conference & OpenIOT Summit North America March 12-14, 2018 -

127. BESANA, Monica and BORGATTI, Michele. *Application Mapping to a Hardware Platform through Automated Code Generation Targeting a RTOS: a Design Case Study,* In: 2003 Design, Automation and Test in Europe Conference and Exhibition. 2003.

128. HOUHOU, Sara, KAHLOUL, Laid, BENHARZALLAH, Saber and BETTIRA, Roufaida. *Framework For Wireless Sensor Networks Code Generation From Formal Specification.* In: Computer Science & Information Technology (CS & IT) Computer Science Conference Proceedings (CSCP) Natarajan Meghanathan et al. (Eds) : NeCoM, SEAS, CMCA, CSITEC – 2017 pp. 35– 52, 2017. © CS & IT-CSCP 2017 DOI : 10.5121/csit.2017.71204

129. WEIGERT, Thomas, WEIL, Frank, VAN-DEN-BERG, Aswin, DIETZ, Paul, and MARTH, Kevin. *Automated Code Generation for Industrial-Strength Systems.* In: Proceedings of the 2008 32nd Annual Ieee International Computer Software and Applications Conference. 2008.

130. CICCOZZI, Federico, CICCHETTI, Antonio, SJÖDIN, Mikael. *Full Code Generation from UML Models for Complex Embedded Systems,* Second International Software Technology Exchange Workshop (STEW) Publisher: Swedsoft. 2012.

131. **BRAGARENCO, A**., RITTMAN, D., SONTEA, V. *Proiectarea Circuitelor Integrate Asistată De Verificarea in Timp Real.* In: Proceedings of the 6th International Conference on Microelectronics and Computer Science. October 1-3, 2009, p.386-389

132. BOUSETTA, Brahim, EL BEGGAR, Omar and GADI, Taoufiq, *Automating Software Development Process: Analysis-PIMs to Design-PIM Model Transformation.* In: International Journal of Software Engineering and Its Applications Vol.7, No.5 (2013), pp.167-196 DOI : 10.14257/ijseia.2013.7.5.17

133. GERHART, Markus, BAYER, Julian, HÖFNER, Jan Moritz and BOGER, Marko. *Approach to Define Highly Scalable Metamodels Based on JSON*. In: Proceedings of the 3rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations 2015 federation of conferences, L'Aquila, Italy, July 23, 2015, pp. 11–20.

134. *Graphviz - Graph Visualization Software.* [Online] Disponibil: https://Graphviz.org/ Disponibil: August 20, 2020.

135. *Layered Software Architecture - AUTOSAR Release 4.2.2.* AUTOSAR Offical Site [online]. [citat 16.06.2020] Disponibil: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

136. ZENG, H., ZENG, H., NATALE, M. D., GIUSTO, P., and SANGIOVANNI-VINCENTELLI, A.. *Stochastic Analysis of CAN-Based Real-Time Automotive Systems.* In: IEEE Transactions on Industrial Informatics, 5(4), 2009. 388–401 p. DOI: 10.1109/TII.2009.2032067

137. YOUNES, M.. *AUTOSAR Application Development* Technische Universitat Chemnitz, In: Seminar Report Nr.: 432674, 03.March 2017.

138. HEINECKE, H., SCHNELLE, K. P., FENNEL, H., BORTOLAZZI, J., LUNDH, L., LEFLOUR, J., MATÉ, J.L., NISHIKAWA, K., and SCHARNHORST, T., *The AUTomotive Open System ARchitecture-An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures.* In: Convergence International Congress & Exposition on Transportation Electronics. 2004.

139. MANNS, P., *Instrumenting AUTOSAR for dependability assessment: A guidance framework.* In: Proc IEEE/IFIP International Conference on Dependable Systems and Networks, 2012.

140. NAVET, Nicolas, SIMONOT-LION, Francoise, *Automotive Embedded Systems Handbook.* Taylor & Francis, pp.1-470, 2008.

141. MARSHALL, Brain, *How Microcontrollers Work,* Founder of HowStuffWorks. (Online): http://electronics.howstuffworks.com/microcontroller.htm, 2014. Disponibil August, 2019.

142. HUANG, Jun, et al. *In-Vehicle Networking: Protocols, Challenges, and Solutions*. IEEE Network 33.1 (2018): 92-98.

143. SIVARAM, P.. *AUTOSAR: In-vehicle Standardization with Certainty of Operations towards Globalization.* In: International Journal of Innovations in Engineering and Technology. 4. 66-71. (2014).

144. ARZEN, Karl-Erik, BICCHI, Antonio, DINI, Gianluca, HAILES, Stephen, JOHANSSON, Karl H., LYGEROS, John, TZES, Anthony. *A Component-Based Approach to the Design of Networked Control Systems*. In: European Journal of Control (13), pp. 261–279, 2007.

145. *Design Patterns Catalogue AUTOSAR CP Release 4.3.0*. AUTOSAR Offical Site [online]. [citat: June 16, 2020]. Disponibil: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_AIDesignPatternsCatalogue.pdf

146. DAFANG, W. and et al.. *Survey of the AUTOSAR Complex Drivers in the Field of Automotive Electronics*. In: Proc. of Int. Conf. on Int. Comp. Tech. and Aut. (ICICTA'2010), Changsha, pp. 662-664, 11-12 May 2010.

147. SANGIOVANNI, A. and NATALE, M. Di. *Embedded System Design for Automotive Applications*. In: IEEE Computer, vol. 40, no. 10, pp. 42-51, 2007.

148. KUM, D., PARK, G. M., LEE, S. and JUNG, W.. *AUTOSAR migration from existing automotive software*. In: Proc. of Int. Conf. on Cont., Aut. and Syst. (ICCAS'2008), Seoul, pp. 558-562, 14-17 Oct. 2008.

149. FRASER, G., WOTAWA, F. and AMMANN, P. E.: *Testing with Model Checkers: A survey.* In: SNA TECHNICAL REPORT, 2007.

150. FRANCO, Felipe, NEME, João Henrique, SANTOS, Max, ROSA, Joao and FABBRO. Inacio. Workflow and toolchain for developing the automotive software according AUTOSAR standard at a Virtual-ECU. 869-875. (2016). DOI :10.1109/ISIE.2016.7745004.

151. TRUMLER, Wolfgang, HELBIG, Markus, PIETZOWSKI, Andreas, SATZGER, Benjamin, UNGERER, Theo. (2007). *Self-configuration and self-healing in AUTOSAR.* In: SAE Technical Papers. 10.4271/2007-01-3507

152. KRAMMER, M., MARTIN, H., KARNER, M., WATZENIG, D. et al., System Modeling for Integration and Test of Safety-Critical Automotive Embedded Systems, In: SAE Technical Paper 2013-01-0189, 2013, https://doi.org/10.4271/2013-01-0189.

153. HELBIG M.: *Selbstkonfiguration in AUTOSAR,* Universität Augsburg, Diplomarbeit, August 2006

154. **BRAGARENCO, Andrei**, MARUSIC, Galina, CIUFUDEAN, Calin. *Layered Architecture Approach of the Sensor Software Component Stack for the Internet of Things Applications*. In: WSEAS Transactions on Computer Research, ISSN / E-ISSN: 1991-8755 / 2415-1521, Volume 7, 2019, Art. #15, pp. 124-135.

155. **BRAGARENCO, Andrei**, MARUSIC, Galina. *Internet of Things System for Environmental Map Acquisition.* In: Journal of Engineering Science, XXVI (4), pp. 88–102. DOI: 10.5281/zenodo.3591594, December 23, 2019

156. **BRAGARENCO** A, MARUSIC G, CIUFUDEAN C. *Communication Chain in the Internet of Things with Spread-out Electronic Device System Abstraction.* In: 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS) (pp. 1-5). IEEE. DOI: 10.1109/IEMTRONICS52119.2021.9422565, Apr 21, 2021, Toronto, ON, Canada

157. LUPAN O., SHISHIYANU, S.,SHISHIYANU T., SONTEA V.,**BRAGARENCO A**.. *Novel zinc oxide nanostructured thin films for volatile organic compounds gas sensors*. In: Proceeding of the International Semiconductor Conference Micro-and Nanotechnologies 29th Edition, September 27-29, 2006, Sinaia, Romania, pp.201-204

158.IAVORSCHI, A., PAHOMI, V., PIRTAC, V., ANGHILOGLU, D., RAILEAN, S., **BRAGARENCO, A**., SCRIPNIC, V.. *Information system analysis of heart rate variability.* In: Proceedings of the International Conference on Nanotechnologies and Biomedical Engineering (ICNBME-2011), Chisinau, Moldova, July 7-8, 2011, pp.445-447. ISBN 978-9975-66-239-0

159.ŞONTEA, V., ARMENCEA, N., ANGHILOGLU, D., **BRAGARENCO, A**., IAVORSCHI, A., PAHOMI, V.. *Sistem de înregistrare şi prelucrare a fotopletismogramelor.* In: Proceedings of The 1st International Conference Radio electronics, informatics, technology, Chisinau, 15-16 October 2008. . pp. 178-183

160.ŞONTEA, V., ARMENCEA, N., **BRAGARENCO, A**., IAVORSCHI, A., PAHOMI, V., ZATUŞEVSCHII, I. *Didpozitiv de înregistrare şi prelucrare a fotopletismogramelor FPG-1.* In: Proceedings of The 2nd International Conference Telecommunications, Electronics and Informatics. Chisinau, 15-18 May 2008. VI pp. 165-168

161.*Engineer's Guide to Signal Conditioning.* National Instrument Oficial Site [Online]. [citat 17.06.2020]. Disponibil: https://www.ni.com/en-us/innovations/white-papers/09/what-is-signal-conditioning-.html.

162.LEEB, Steven B., ORTIZ, Alfredo, LEPARD, Robert F., SHAW, Steven R., and KIRTLEY, James L.Jr.. *Applications of Real-Time Median Filtering with Fast Digital and Analog Sorters* , In: IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 2, NO. 2, JUNE 1997

163.*Technopedia - Actuator*. Technical Enciplopedia [online]. [citat 17.06.2020]. Disponibil: https://www.techopedia.com/definition/17043/actuator

164.**BRAGARENCO, A**., TEODORESCU, N., ŞONTEA, V.. *Sistem de achiziţie a semnalelor biomedicale.* In: Proceedings of the 5 th International Conference on Microelectronics and Computers Science, Chişinau 2007 V.II pp. 34-38

165.ARMENCEA, N., ŞONTEA, V., **BRAGARENCO, A**., ANGHIOGLU, D., PAHOMI, V. *Dispozitiv pentru înregistrarea şi prelucrarea fotopletismogramelor.* In: Proceedings of the 5th International Conference on Microelectronics and Computers Science, Chişinau 2007 V.II pp. 24-27

166.**BRAGARENCO, A**., SONTEA, V.. *Evaluation and Control System for a Laboratory Experiment.* In: Proceeding of European Conference on the Use of Modern Information and Communication Technologies 2006, Gent, Belgium, 30-31 March 2006, p.362 – 366

167. **BRAGARENCO, A**., PLOTNIC, P., SONTEA, V. *Gas sensors research and control system.* In: Proceeding of the 4th International Conference on Microelectronics and Computer Science. Chişinău, 2005, v.2, pp.443-446

168. **BRAGARENCO, Andrei**. *Sensor-Actuator Software Component Stack for Industrial Internet of Things Applications.* 24th International Conference on System Theory, Control and Computing, pp. 540-545, October 8 - 10, 2020, Sinaia, Romania.

169. TROCIN, Caterina, **BRAGARENCO, Andrei**, SONTEA, Victor. *Expert system for Medical Diagnosis. Fuzzy Logic.* In: Proceedings IIS - International Workshop on Intelligent Information Systems. Institutul de matematică şi Informatică, Chisinau, Moldova, 13-14 Septembrie 2011, pp. 207-210. ISBN 978-9975-4237-0-0

170. **BRAGARENCO, Andrei**. *Method for Embedded Systems Development by Configurations and Code Generation Based on Json Metamodels.* In: Revista de Ştiinţă, Inovare, Cultură şi Artă „Akademos" 3(58) / 2020 pp 19-27, ISSN 1857-0461 DOI: 10.5281/zenodo.4269373

171. BRÄUNL, Thomas. *Embedded Robotics.* Springer, Berlin, Heidelberg , (2003) ISBN 978-3-662-05101-6

172. *A Guide to Fault Detection and Diagnosis.* Performity LLC/Greg Stanley and Associates [online]. [citat 20.06.2020]. Disponibil: - https://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/faultdiagnosis.htm

173. RIVERA-GUILLEN, J.R., ROMERO-TRONCOSO, R.d.J., OSORNIO-RIOS, R.A., GARCIA-PEREZ, A. and HERRERA-RUIZ, G.. *Design methodology for fully dynamic-controlled polynomial profiles and reduced tracking error in CNC machines.* In: Int J Adv Manuf Technol 51, pp. 723–737, 2010. DOI: 10.1007/s00170-010-2650-2

174. BOSCARIOL, P., and GASPARETTO, A. *Model-based trajectory planning for flexible-link mechanisms with bounded jerk*. In: Robotics and Computer Integrated Manufacturing. Volume 29, Issue 4, pp. 90-99, August 2013. DOI: 10.1016/j.rcim.2012.11.003

175. *Comparison of Different Motion Types.* Robotic Systems Integration Site [online] [citat 20.06.2020]. Disponibil: https://docs.roboticsys.com/rmp/topics/motion/motion-concepts/comparison-of-different-motion-types

176. LUAY, Alawneh, and HAMOU-LHADJ, Abdelwahab. *Pattern Recognition Techniques Applied to the Abstraction of Traces of Inter-Process Communication*. In: 2011 15th European Conference on Software Maintenance and Reengineering, 2011. doi:10.1109/CSMR.2011.27.

177. MAMIDALA, Amith R. *Architecture of the Component Collective Messaging Interface*. In: International Journal of High Performance Computing Applications, 2010.

178. *Organizational Behavior / Human Relations, ch 8, Key Components of Communication*. Learneo - Course Sidekick site [online], [citat 20.06.2020]. Disponibil: https://courses.lumenlearning.com/wmopen-organizationalbehavior/chapter/key-components-of-communication/

179. *AUTOSAR Release 4.2.2. Specification of DIO Driver.* Autosar official Site [online]. [citat: 16.06.2020]. Disponibil: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_SWS_DIODriver.pdf

ANEXE

Anexa 1. Considerente de proiectare a componentelor software la nivelul aplicației

Aplicând metodologia expusă în teză, efortul principal îi revine dezvoltării aplicației conform specificației sistemului prin definirea funcționalităților componentelor funcționale sau comportamentale al sistemului. În aceasta anexa este prezentat procesul de proiectare a sistemului de control al unui frigider inteligent făcând abstracție de platforma care realizează interacțiunea cu mediul extern.

Frigiderul inteligent este un electrocasnic care este programat să detecteze ce tipuri de produse sunt depozitate în interiorul acestuia și ține evidența stocului prin scanarea codurilor de bare sau RFID. Frigiderul este echipat să se determine singur ori de câte ori un produs alimentar trebuie completat. Controlerul frigiderului este un modul hardware pentru achiziția semnalului și generarea de semnale de control pentru părțile electromecanice ale frigiderului, precum și pentru alte funcții precum controlul luminii interioare, soneria și alte caracteristici electrice Fig. A1.1.

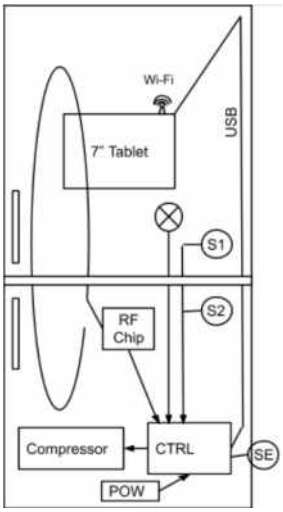


Fig. A1.1. Structura de ansamblu al frigiderului Inteligent

Întreg sistemul este compus din doua pârți componente principale:

- Sistem incorporat de control a frigiderului pentru controlul funcționalităților de bază cum ar fi temperatura, umiditate, protecții motoare, gestionarea iluminării interne, etc.
- Sistem multimedia pentru gestionare funcționalităților de frigider inteligent în mare orientate pentru managementul produselor alimentare.

Sistemul electronic incorporat sau ECU gestionează un frigider cu două camere cu un singur compresor: Camera frigorifica este situata în partea superioara iar congelatorul este situat în partea de jos.

Unitatea electronică de control (ECU) este proiectată pentru a îndeplini următoarele funcții - activarea, dezactivarea și indicarea includerii în rețea; Controlul temperaturii în camera frigorifică; raportarea temperaturii setate, raportarea modurilor de funcționare suplimentare. Control frigiderului pe lângă comportamentul general al frigiderului include funcționalități de gestionare a modulelor în funcție de starea de eroare, gestionarea energiei pentru salvare stare în EEPROM la oprire, activare compresorul numai dacă timpul de oprire a motorului este verificat de RTC, detectează situația de putere scăzută și dezactivează compresorul daca e cazul, salvarea timpului de oprire curent a extras din RTC și salvarea în EEPROM atunci când este detectată oprirea motorului; inițiază dialog prin canalul de comunicare în situații critice, trimițând mesaje de eroare sau de avertizare.

In regim normal de funcționare comportamentul frigiderului va fi după cum urmează. La pornire, un led indicator va fi aprins iar la detectarea unei erori, va clipi și va raporta numărul de eroare după numărul de clipire.

Arhitectura software încorporat

In aceasta anexa este descrisă proiectarea arhitecturala de sistem pentru componentele de control al frigiderului care pune la dispoziție sistemului multimedia diverse servicii de control al frigiderului pentru a satisface cerințele sistemului de management al produselor alimentare. Arhitectura generala a întregului sistem este prezentată în diagrama arhitecturala din Fig. A1.2. Modulul multimedia ce implementează „Inteligenta" frigiderului, este prezentată în următoarea anexă.

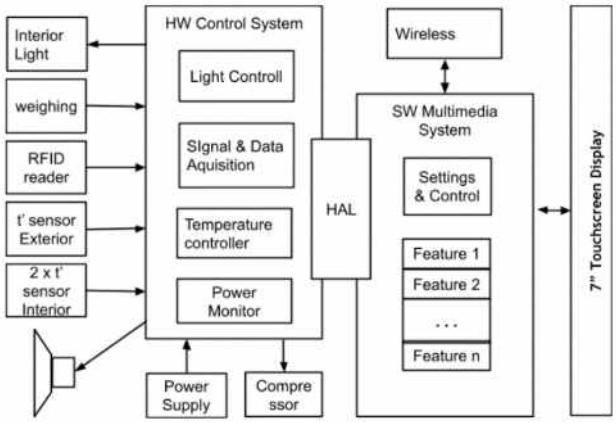


Fig. A1.2. Arhitectura de sistem pentru al Frigiderului Inteligent

Sistemul de control al funcțiilor de baza al frigiderului reprezintă un set de componente ce pot fi regăsite întru-un frigider contemporan clasic și consta dintr-un set de componente cum ar fi iluminare interna, control deschidere ușă, control motor al pompei de căldură, control de

temperatură, alarma sau sonerie, ș.a. O prezentare generala al arhitecturii de sistem al sistemului de control al frigiderului este prezentată în Fig. A1.3.

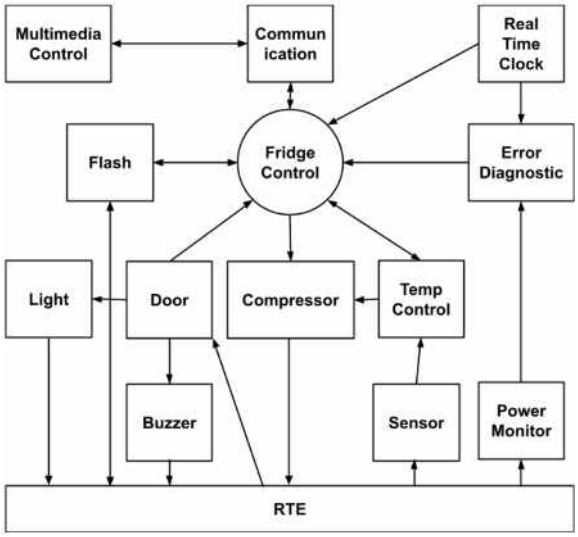


Fig. A1.3. Arhitectura sistemului de control al frigiderului

Componente de interacțiune cu mediul

Modelarea acestor componente este una clasică și se supune conceptului de modelare a componentelor generice în straturi prezentate în teză, ca regula aceste componente conform diagramei expuse mai sus sunt reprezentate de interfața RTE care pune la dispoziție serviciile furnizate de aceste componente.. Pentru interacționarea cu mediu extern sistemul este dotat cu următoarele componente de interfață:

- Senzor de temperatura interioara - Doi senzori de temperatură interioară, câte unul pentru fiecare cameră. Starea senzorilor este scanată continuu, iar valoarea curentă va fi folosită pentru monitorizare și controlul temperaturii din interiorul frigiderului.
- Senzor de temperatura ambientala - starea temperaturii exterioare este folosită pentru afișarea pe ecranul frigiderului, tipul de senzor este același cu senzorii interiori
- Buzer - un modul care emite semnale sonore pentru a informa și/sau avertiza utilizatorul. această caracteristică ar putea fi implementată ca un sonerie separată sau integrată în sistemul multimedia, folosind difuzoare.
- Led Indicator alimentare electrica - indicator cu semnal pentru a informa sau avertiza utilizatorul. funcționalitate similară cu cea a controlului luminii interioare.
- Senzor de detectare deschidere ușă – este realizat prin tehnologia hall sau un simplu buton, pe ieșire generând un semnal binare de ușă închisa sau deschisă.
- Senzor sursa electrică - un modul de alimentare cu interfață de monitorizare a puterii. Funcția de monitorizare ar putea fi utilizată pentru selectarea unei strategii de gestionare a energiei.

- Actuator - interfață de ieșire pentru a controla compresorul în regim de ON-OFF.
- Interfața USB – interfață realizează interacțiunea sistemului de control cu sistemul multimedia al frigiderului inteligent.

Proiectare sistem de operare

Pentru asigurarea execuției sarcinilor din cadrul componentelor sistemului este elaborate un sistem simplu de operare, care permite gestionarea și asigurarea execuției mai multor sarcini în paralel. Diagrama conceptuala a acestui sistem de operare este prezentata în Fig. A1.4.

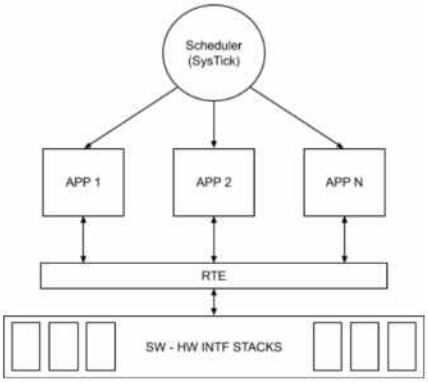


Fig. A1.4. Diagrama conceptuala a sistemului de operare

Sistemul de operare este unul non-preemptiv. Fiecare componenta are o funcție de sarcină și o structură de descriptor unde structura are un pointer către funcția prin care trebuie apelată. Conform structurii descriptorului, microcontrolerul prin intermediul unui timer se sistem apelează funcția de sarcină. Toate sarcinile sunt înregistrate printr-o serie de descriptori de sarcini. *Scheduler* selectează în funcție de prioritate și periodicitate ce sarcină va trebui să fie selectată pentru execuție în intervalul de timp curent numit *tick*, și ordinea execuției, unde un *tick* este perioada dintre două întreruperi ale unui timer de sistem. Funcția-sarcină este considerată că nu necesită timp, deoarece nu conține întârzieri în interior. Execuția sarcinii care depinde de timp și întârzierile procesului vor fi proiectate conform principiului FSM secvențial. Funcția de sarcină verifică dacă are vreun mesaj, după ce trece la evaluarea corpului său (FSM sau altele). Întârzierile sunt implementate prin setarea perioadei de întârziere și așteptarea până când întârzierea este terminată caracteristicile din structura descriptorului. *Scheduler* verifică dacă vreo aplicație are mesaje sau este întârziată. Accesul la modulele periferice MCU este realizat printr-un HAL. Întârzierile sunt implementate de câmpul time-out din descriptorul de sarcină. planificatorul numără invers toate variabilele time-out. valoarea 0 înseamnă că expirarea este atinsă și nu se mai numără. Planificatorul indică la descriptorul de sarcină curent și apelează funcția din acesta prin referință. La o eroare critică, un mesaj este setat pentru toate sarcinile, astfel încât fiecare sarcină

ar putea reacționa corect la semnal. Principiul round-robin este aplicat pentru aceleași sarcini prioritare.

Proiectarea sistemelor de Control.

Componentele de control al sistemului ca regula sunt specifice fiecărei aplicații și sunt proiectate conform cerințelor specifice fața de sistem. Conform arhitecturii de sistem al sistemului de control al frigiderului, Fig. A1.5, fiecare dintre componente implementează o funcționalitate speciala, aceste fiind descrise în continuare. În diagramele condiționale din această anexa ramura din stânga se considera FALS iar din dreapta ADEVARAT. În cazurile când este diferit, este specificat cu notații caracteristice.

Controlul și protecția motorului compresor

Motorul compresor pentru pompa de căldură frigiderului este componenta principala pentru realizarea funcției principale a frigiderului, și anume a răcirii aerului în camera frigorifice. Funcționarea continua a acestuia ar putea duce la ieșirea din funcție a lui. În acest scop s-a realizat un algoritm de protecție a acestuia conform cerințelor de la client după cum urmează:

- Compresorul trebuie oprit atunci când: tensiunea scade la 170 V ± 5 V timp de 1 min, la tensiuni peste 260 V ± 5 V timp de 100 ms, în caz de întrerupere a curentului electric.
- Când rețeaua de alimentare este restabilită la valori de tensiune în intervalul de la 187 la 265 V, compresorul pompei de căldură trebuie pornit nu mai devreme de 3 minute. după oprire (dacă pauza de lucru a fost scurtă). ECU transferă activitatea pompei de căldura la regimul de temperatură care a fost setat înainte ca motorul să fie oprit.
- Disponibilitatea pompei de căldură este semnalizata printr-un semnal luminos.

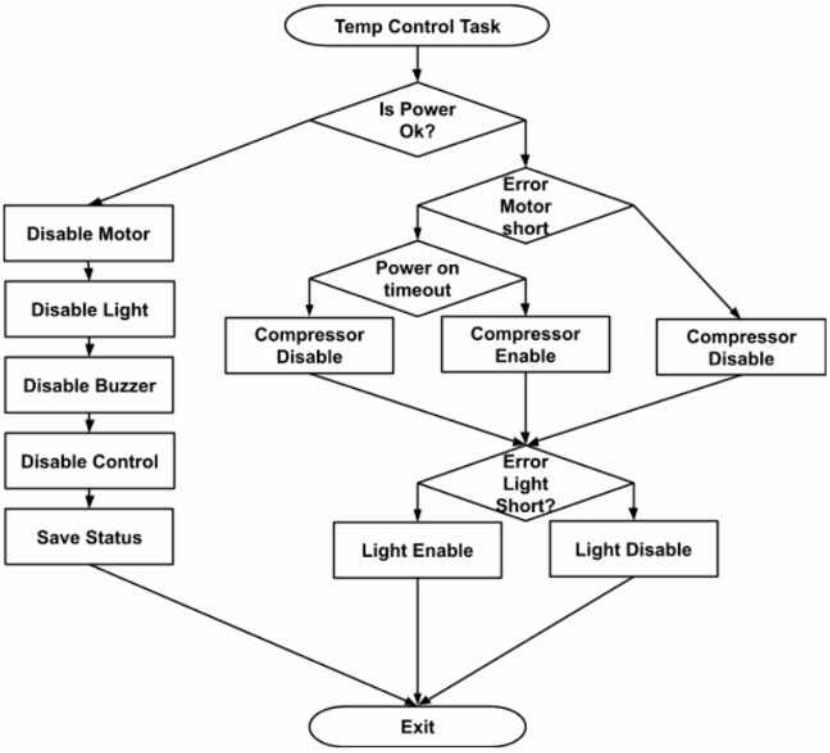


Fig. A1.5. Algoritmul de protecție al motorului pompei de căldură

In aceasta diagrama *"Is Power ok?"* reprezintă verificarea când rețeaua este restabilită, *"error motor short"* reprezintă verificarea daca tensiunea este suficienta pentru operare, *"Power on timeout"* reprezintă verificarea timpului de după oprirea motorului iar *"Error Light Short?"* este verificarea erorii pe alimentarea compresorului. *"Compressor Enable"* și *"Compressor Disable"* reprezintă disponibilitatea compresorului pentru operare, iar *"Light enable"* și *"Light disable"* – indicatorul de eroare pe alimentarea compresorului.

Controlul compresorului

Controlul compresorului reprezintă activarea pompei de căldura prin interpretarea cererilor de la componenta de control al temperaturii. Implementarea acestui comportament este realizată conform diagramei din Fig. A1.6, conform căreia motorul compresorului va putea fi pornit doar în cazul daca va avea permisiune de la sistemul de protecție, prin semnalul de disponibilitatea compresorului.

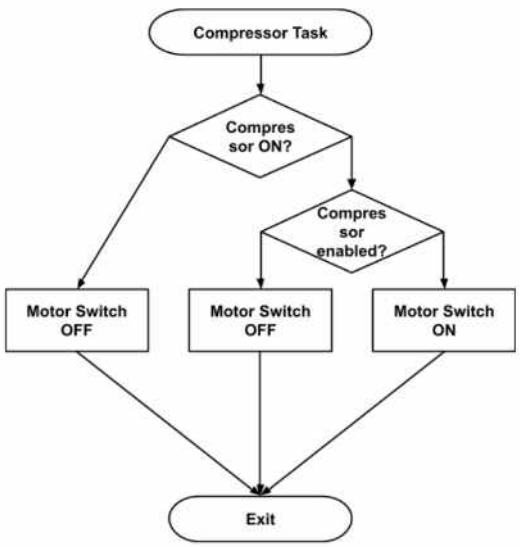


Fig. A1.6. Algoritmul de activare a motorului pompei de căldură

Unde *"Compressor ON?"* este cererea de activare a compresorului, *"Compressor Enabled?"* disponibilitatea compresorului, "Motor Switch ON " și *"Motor Switch OFF"* sunt comenzile de activare a motorului.

Controlul temperaturii

Control temperaturii în camera frigorifica este realizată cu un sistem de control ON-OFF cu histerezis, conform unei strategii pentru optimizarea consumului de energie definita de către client, care pornește și oprește compresorul utilizând temperatura definită de utilizator conform cu Tabelul A4.1 pentru valorile cerute de utilizator. Pentru cererile intermediare se aplica interpolare lineara.

Tabelul A4.1Tabel cu valorile temperaturii de control

Temperatura setata	Temperatura ON	temperatura OFF
+8	+10	+7
+7	+9	+6
+6	+8	+5
+5	+7	+4
+4	+6	+3
+3	+5	+2
+2	+4	+1

Algoritmul de control al temperaturii se va implementa conform diagramei din Fig. A1.7. Algoritmul de control ON-OFF cu histerezis, și va implementa controlul ON-OF cu histerezis.

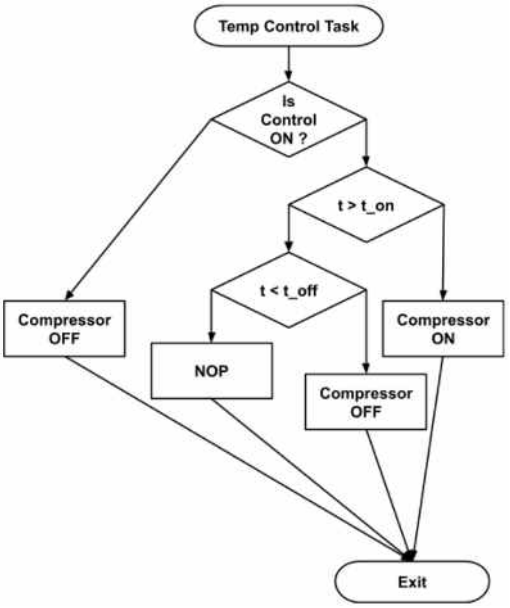


Fig. A1.7. Algoritmul de control ON-OFF cu histerezis

Unde *"Is Control ON?"*este semnalul de activare a controlului de temperatura, *"Compressor ON"* și *"Compressor OFF"* comenzile de activare și dezactivare de a cererii către compresor.

Controlul luminii interioare

Pentru diminuarea pierderilor de răcire, frigiderul este dotat cu un sistem de monitorizare a deschiderii ușii, cu asigurarea iluminării în interior – se va aprinde lumina interioară când ușa frigiderului este deschisă. și se va stinge lumina interioară când ușa frigiderului este închisă. Totodată frigiderul este dotat cu notificare sonora în cazul când ușa a este deschisa mai mult decât limita presetată. Acest comportament este descris în diagrama din Fig. A1.8.

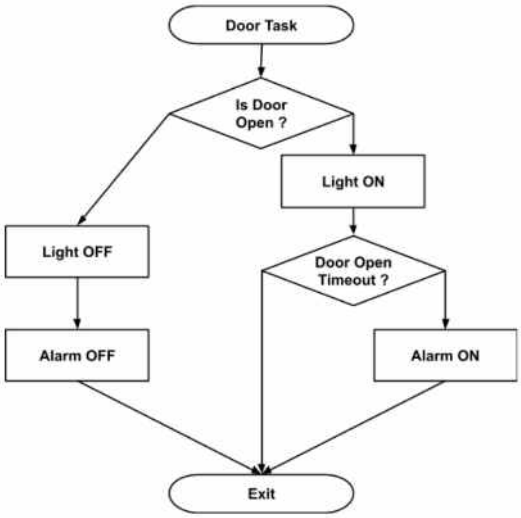


Fig. A1.8. Algoritmul de monitorizare a ușii frigiderului

Unde *"Is door open?"* este semnalul de detecție a închiderii ușii, *"Light ON"* comanda de aprindere lumina interioara, *"Light OFF"* comanda de stingere a luminii, *"Alarm ON"* comanda de pornire a alarmei, *"Alarm OFF"* comanda de oprire a alarmei sonore.

Controlul luminii în camera frigiderului este realizata conform diagramei din Fig. A1.9.

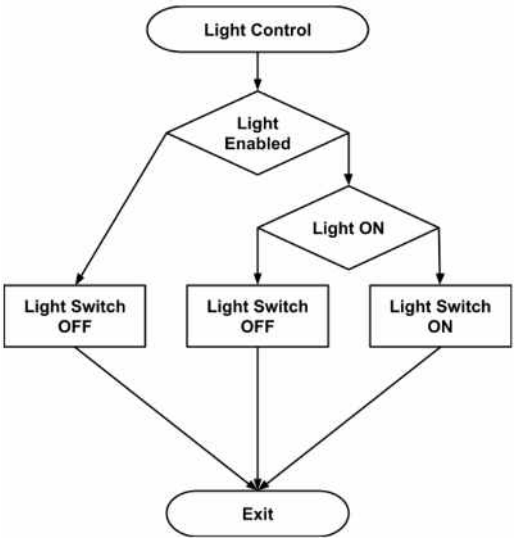


Fig. A1.9. Algoritmul de control al luminii în camera frigiderului

Unde *"Light Enabled"* este configurația generală de activare a aprinderii luminii când ușa este deschisa, *"Light ON"* – semnalul de cerere *aprindere* lumina, iar *"Light Switch ON"* și *"Light Switch OFF"* sunt comenzile de aprindere a corpului luminos pentru iluminarea camerei frigiderului.

Control notificare sonoră – Buzer

In scopul notificării sonore utilizatorului sistemul este dotat cu o alarma sonora care în cazul monitorizării ușii va fi activata după cum urmează - se va contoriza timpul în care ușa frigiderului este deschisă. Dacă timpul de deschidere a ușii aste mai mare decât pragul de timp admisibil de deschidere a ușii, se va activa un semnal sonor cu o perioadă de 2 secunde (1 secundă ON/1 secundă OFF). Acest comportament este descris în diagrama din Fig. A1.10.

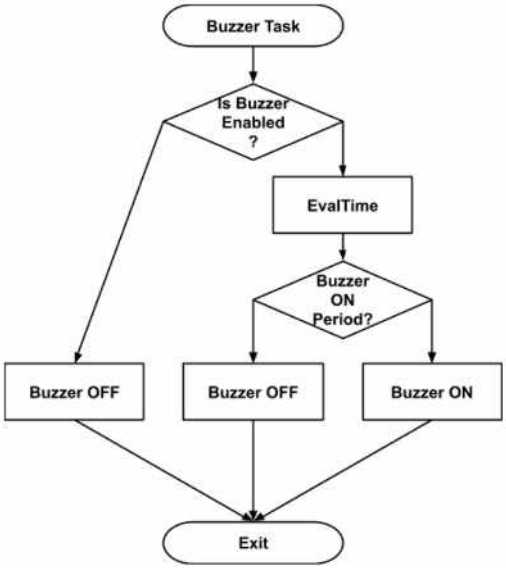


Fig. A1.10. Algoritm de notificare sonoră

Unde *"Is Buzzer Enabled?"* este semnalul de activare al alarmei, *"EvalTime"* secțiunea de contorizare a timpului pentru formarea mesajului sonor cu un raport Activ/pasiv, *"BuzzerOnPeriod?"* semnal ce indica perioada activă de emitere a sunetului *"Buzzer ON"* și *"Buzer OFF"* – comanda de activare sau dezactivare a emiterii semnalului sonor.

Comunicarea prin interfața USB.

Sistemul electronic incorporat de control al frigiderului inteligent este dotat cu un modul de comunicare prin interfața USB. Modulul de comunicare permite interacțiunea cu echipamente externe în scop de monitorizarea a stării interne, dar și a controlului la distanță a sistemului electronic incorporat prin interfața USB. Procesul de comunicare este realizat schimbul de date împachetate conform unui protocol de comunica specific aplicației ace permite citirea parametrilor colectați de la senzori, transmiterea de configurațiilor de control al frigiderului, cat și raportarea erorilor de funcționare. Formatul pachetelor de raportare a stării curente a senzorilor este prezentat în Fig. A1.11.

Nr. pachetului	Timpul de sincronizare.	Senzor Current	Senzor Tensiune	Senzor Putere	Sensori Temperatură	Returnarea starii de control

Fig. A1.11. Pachet de transmitere a datelor de pe senzori.

Modulul de comunicare prin interfața USB va scana în continuu pentru detectarea pachetelor de intrare. În cazul recepționarii reușite, va decodifica pachetul interpretând comanda conform conținutului acestuia și va răspunde cu starea curentă sistemului cu un conținut de valori de pe senzori, și starea de execuție a comenzii recepționate. În caz de erori de comunicare sau executare , va răspunde cu un pachet de eroare. Comportamentul modulului de comunicare este prezentat în Fig. A1.12.

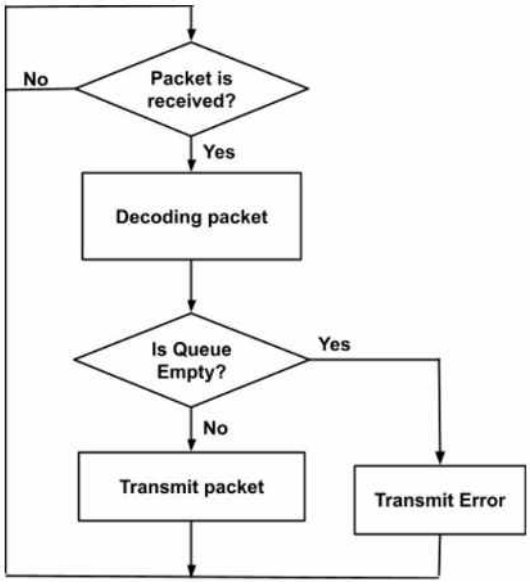


Fig. A1.12. Modul de funcționare a modului de comunicare prin USB

Anexa 2. Considerente de proiectare a componentelor electrice

Conform principiilor de modelare a sistemelor electronice distribuite, luând în considerație arhitectura pe nivele, de exemplu cazul uscătoriei de fructe prezentate în Fig. 3.51, componentele hardware presupun o proiectare în domeniul ingineriei electrice, care ca regula sunt proiectate specific pentru fiecare dispozitiv electronic.

Chiar daca nu este posibilă o proiectare complet automatizată, în cazul componentelor realizate în domeniul ingineriei electrice se interveni cu recomandări de abordare reieșind din experiența acumulata pe alte proiecte. Pentru proiectul uscătoriei de fructe considerentele de proiectare electrica sunt prezentate în secțiunile ce urmează.

Intrări-iesiri logice

Modulului Electronic de Control este dotat cu intrări și ieșiri logice, care acționează la nivelul de 24 V curent continuu. Astfel semnalele de intrare trebuie să fie tensiuni de 24 V. Ieșirile sunt le fel tensiuni de 24 V, ce se aplică la releele de dirijare cu rotirea tijei de împingere a stelajelor cu fructe, aprinderea lămpilor indicatoare necesare semnalizării anumitor evenimente operatorului, ș.a. modul de interactiune cu semnale logice este realizat în Fig. A2.1.

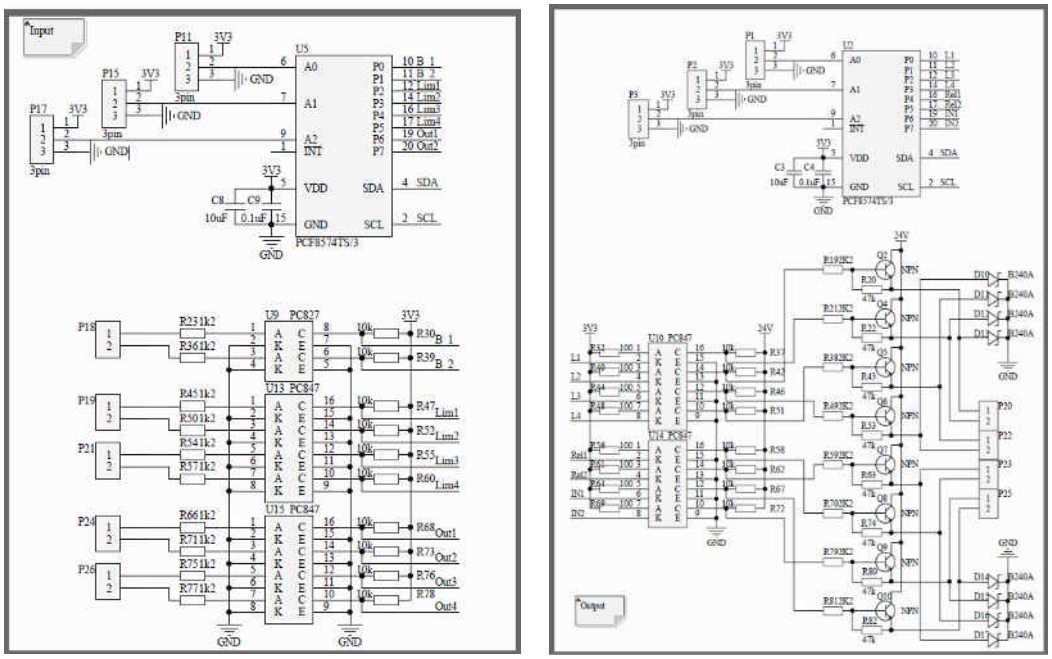


Fig. A2.1. Interacțiuni cu semnale digitale. la nivelul de 24 V curent continuu

Interactiunea cu utilizatorul

Modulul Electronic de control fiind echipament central, de baza, realizează interfața cu utilizatorul, măsurarea parametrilor de temperatură și umiditate a aerului la intrare și ieșire din

camera de uscare, dirijarea și citirea poziției clapetelor de aer, citirea limitatoarelor de cursă și comandarea tijei de împingere a stelajelor cu fructe în camera de uscare, interacționarea cu cazanul.

Interacțiunea cu utilizatorul cuprinde interacțiuni prin intermediul unui afișor LCD, un set de leduri indicatoare conectați la pinii logici, manipulator rotitor bazate pe un encodor și doua butoane. De asemenea este posibila emiterea de semnale sonore ca și notificări în cazul anumitor evenimente sau erori. Parte HW se poate vedea pe Fig. A2.2.

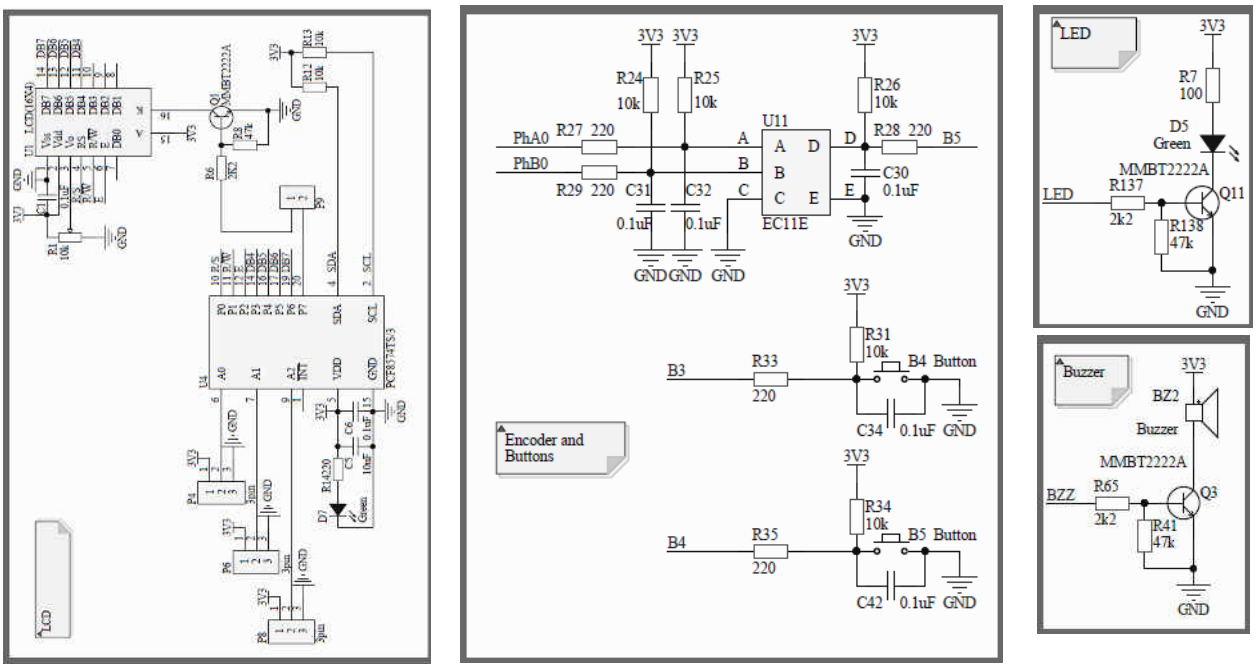


Fig. A2.2. Schema electrica ce realizează interacțiunea cu utilizatorul

Pe Panoul Electric sunt plasate butoanele Start și Stop ale ciclului de acționare a motorului tijei de împingere a stelajelor cu fructe. Aceste butoane sunt dublate de alte două butoane corespunzătoare, plasate în preajma ușii uscătoriei de fructe, pentru a facilita lucrul operatorului. Butoanele Start au contactele Normal Deschise - conectate în paralel, iar butoanele Stop sunt cu contacte de tipul Normal Închis - în serie. La aceste butoane se aplică nivelul de tensiune de 24 V de la blocul de alimentare, iar de la ieșirea acestora la intrările logice ale Blocului Electronic de Control.

Stările limitatoarelor de cursă indicatoare a cărucioarelor la intrare și ieșire din camera de uscare sunt reflectate pe Panoul Electric prin intermediul a două lămpi indicatoare de culoare galbenă. O lampă indicatoare de culoare roșie atenționează operatorul asupra apariției anumitor mesaje de eroare în procesul de uscare și funcționare a uscătoriei de fructe.

Senzori parametri de mediu al instalației de uscare -Temperature Humidity Anemomeeter

Modulul Electronic de Control este dotat cu ieșiri și intrări analogice pentru comanda servomotoarelor clapetelor de aer, citirea unghiului curent de înclinare a clapetelor, valorile temperaturii, umidității aerului în timpul procesului de uscare a fructelor.

Senzorii de temperatură și umiditate a aerului sunt plasați la intrarea aerului în camera de uscare și la ieșirea aerului din aceasta. Datele obținute vor oferi informații valoroase despre procesul de uscare. Senzorii de temperatură selectați sunt durabili, pe bază de Cupru, își modifică rezistența liniar în dependență de temperatura mediului în care se află. Senzorii de umiditate sunt cu ieșire de curent, facilitând citirea datelor. Tensiunea de alimentare este cuprinsă în diapazonul 9 ... 28 V curent continuu, iar curentul de ieșire variază liniar între 4 și 20 mA, 20 mA corespunzător umidității relative a aerului de 100 %. Interacțiunea electrica cu senzorii ce realizează condiționarea semnalelor în domeniul HW este prezentata în Fig. A2.3

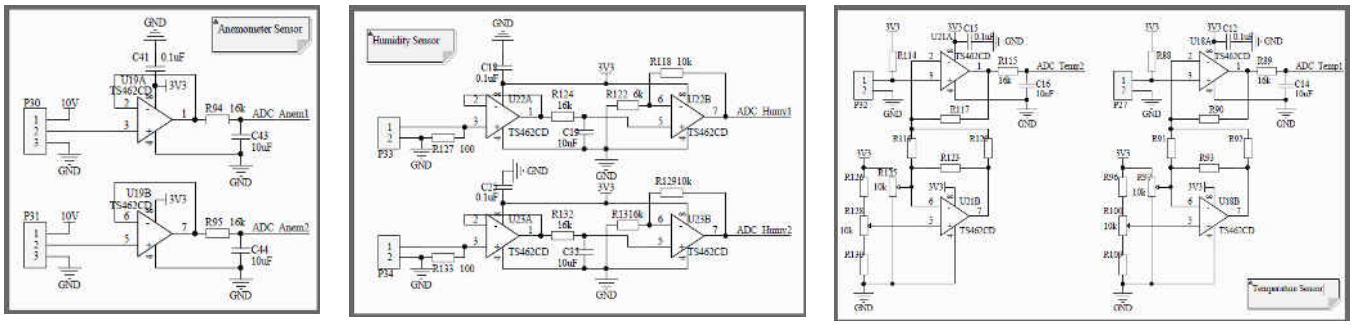


Fig. A2.3. Schema electrica ce realizează interacțiunea cu seniorii

Actuator Servo Motor

Servomotorul clapetei de aer este unul cu comandă analogică a unghiului de deschidere, se alimentează cu tensiunea de 24 V curent continuu, cu contact pentru comanda unghiului de deschidere. La acesta se aplică o tensiune de la 0 la 10 V, ceea ce îl face să modifice unghiul de deschidere a clapetei de aer de la 0 la 90°. Servomotoarele selectate au un semnal analogic destinat indicației unghiului de înclinare a clapetei, ne oferă o tensiune de la 2 la 10 V liniar dependentă de unghiul real, Fig. A2.4. Servomotoarele sunt dotate cu butoane speciale, pentru a oferi posibilitatea modificării manuale a unghiului de deschidere a clapetei.

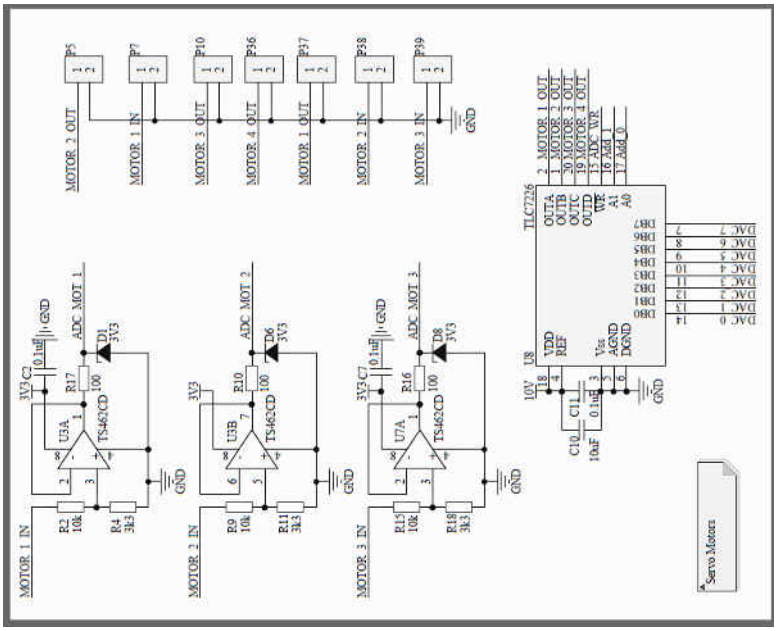


Fig. A2.4. Schema electrica ce realizează interacțiunea cu servo-motorul

Comunicatii Bluetooth și Serial USB

Pentru a avea posibilitate de a adăuga extensii a sistemului de control cu scop de diagnoza și dezvoltarea de aplicații externe de monitorizare și control, sistemul este dotat cu interfețe de comunicare – USB pentru comunicații cu fir, și Bluetooth, pentru comunicații fără fir. Un exemplu de aplicație de control ar fi Aplicații pe telefon mobil, Android, de control a instalației de uscare. Electric interfețele sunt realizate după cum est arătat în Fig. A2.5.

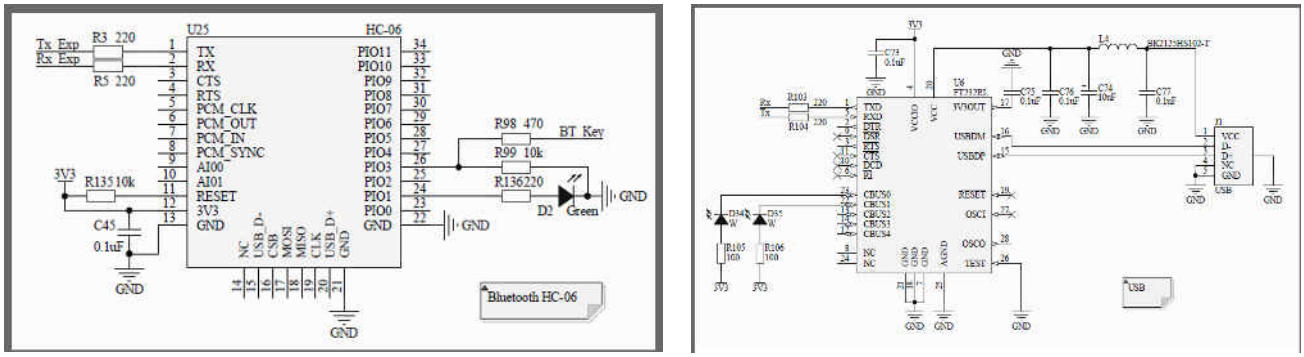


Fig. A2.5. Interfețele de comunicare cu instalația de uscare

Stocare date - SD Card

Pentru a asigura stocarea datelor de configurare a sistemului cât și colectării și stocării informațiilor pe termen lung, în diverse scopiri, sistemul este dotat cu o interfață specializata ce permite atașarea unui stick de memorie flash de tip SD card. Schema de conexiune este reprezentata în Fig. A2.6.

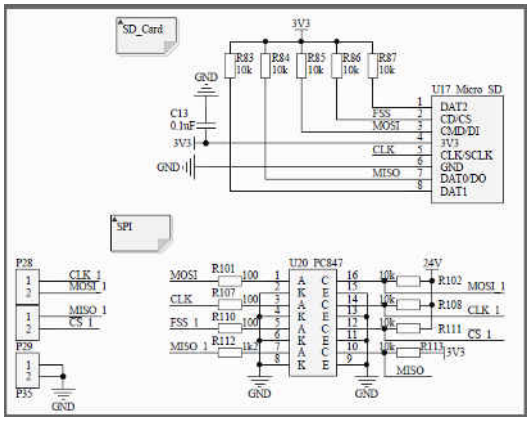


Fig. A2.6. Schema electrica ce realizează conectarea memoriei externe

Unitatea de procesare pentru rularea componentelor resurse de program

Pentru avea posibilitatea de a rula componente realizate prin tehnologii resurse de program este necesar suport electronic capabil sa realizeze aceasta funcție. În scopul proiectului a fost selectat microcontrolerul MCU TM4C123G capabil sa ruleze toate funcționalitățile din componentele realizate ca resurse de program, asigurând performanța din punct de vedere instrucțiuni pe unitate de timp, cât și a capacitații de structuri de date necesare funcționalităților. Schema de încadrare electrica a microcontrolerului în soluție este prezentată în Fig. A2.7.

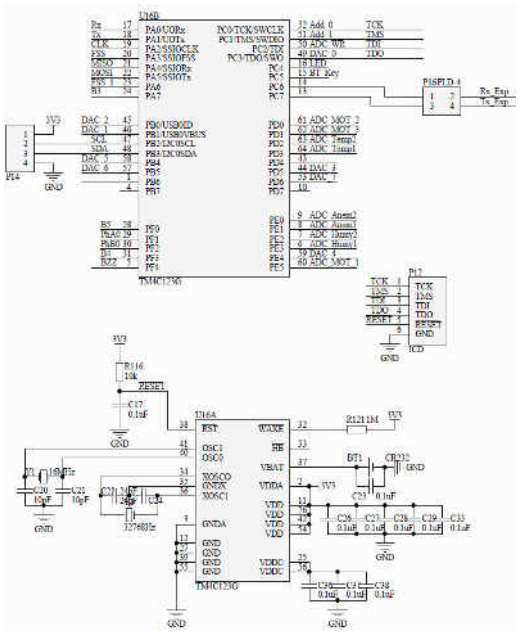


Fig. A2.7. Schema electrica de conexiune a microcontrolerului TM4C123G

Sursa de alimentare

Sursa de alimentare reprezintă o componenta indispensabilă în asigurarea funcționalității dispozitivului. Din punct de vedere funcțional reprezintă o componenta auxiliară în cazul în care se realizează monitorizarea și controlul energiei consumate. Soluția din acest proiect e limitata

stabilizarea tensiunii de alimentare pentru componentele electrice și este realizata excesiv la nivel electric. Schema electrica a soluției este prezentată în Fig. A2.8.

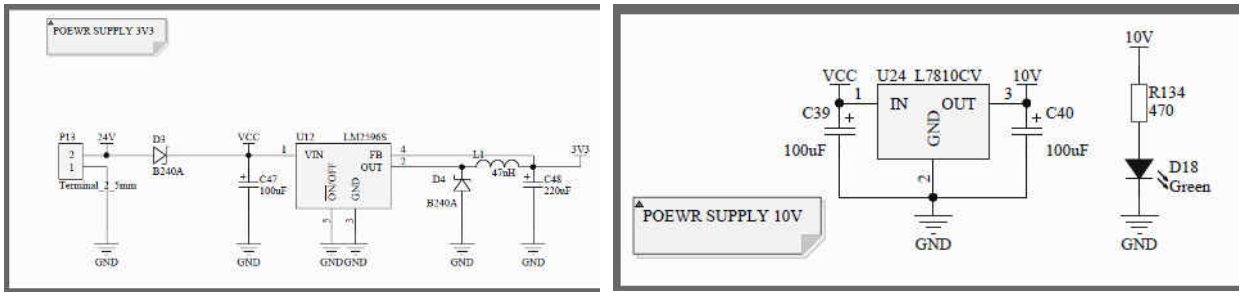


Fig. A2.8. Schema electrica de asigurare a alimentarii cu curent al modulului de control.

Integrarea componentelor electrice în soluție.

În scopul asigurării funcționalității stabile modulului de control pe termen îndelungat a fost realizat un cablaj imprimat de calitate la nivel industrial vederea 3D a căruia este prezentata în Fig. A2.9.

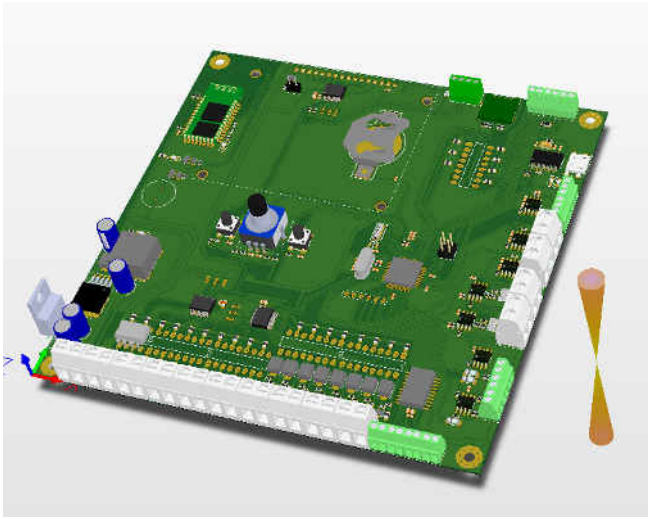


Fig. A2.9. Vedere 3D al cablajului imprimat al modulului de control.

Anexa 3. Considerente de modelare matematică în sistemele electronice distribuite

Cerințele tehnice față de sistemul de direcționării luminii solare

Proiectarea sistemului electronic de orientare a luminii reflectate către țintă s-a realizat ținând cont de cerința de baza din partea comanditarilor și anume - oglinzile amplasate cele 4 colțuri ale clădirii trebuie să reflecte lumina întotdeauna (dacă este posibil din punct de vedere fizic) pe aceeași țintă, amplasata pe centru, în limitele preciziei acesteia. Vizual construcția fizică a sistemului în ansamblu se poate vedea din Fig. A3.1 și Fig. A3.2.

Cerințe generale de sistem

Reieșind din cerințele de mai sus au fost definite cerințe față de sistem pentru asigurarea funcționarii normale și a configurării sistemului după cum urmează:

- Sistemul realizează controlul automat de urmărire solară și calculului poziționării oglinzilor conform modeleului matematic prezentat anterior reieșind din poziția curenta a soarelui și a poziției ținta.
- Sistemul este dotat cu protecție împotriva vântului și a altor perturbări care pot cauza funcționarea defectuoasă a oglinzilor.
- Configurarea parametrilor sistemului se realizează prin interfață terminală wireless al sistemului, printre care: poziție geografică - latitudine și longitudine; declinație magnetică - necesită pentru determinarea corectă a polului nord geografic; oră locală și compensare GMT; control manual al poziției oglinzii pentru setarea punctului țintă; alte calibrări necesare pentru adaptarea sistemului.
- Opțional poate fi inclusă o aplicație GUI pe Android sau PC pentru o modalitate mai intuitivă de configurare a sistemului prin traducerea acțiunii GUI în comenzile terminalului care vor fi trimise prin interfața serială.

Cerinte mecanice față de sistem

Din punct de vedere mecanic construcția sistemului are următoarele constrângeri:

- Amplasarea a patru oglinzi la colturile la distanța de 12 m una față de alta pe acoperișul clădirii conform Fig. A3.1.

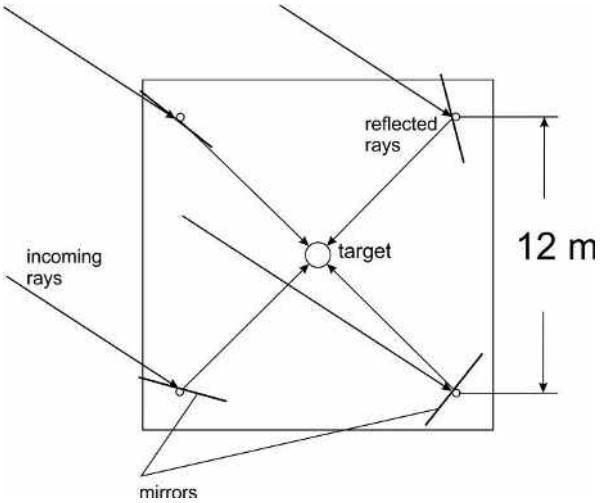


Fig. A3.1. Schița de amplasare a oglinzilor pe acoperișul clădirii

- gestionarea cu ușurință greutatea oglinzilor de 120 kg în oricare dintre configurațiile sale pe două axe de rotație, astfel încât oglinda să poată realiza orice configurație, pentru orice poziție a soarelui lumina să fie reflectată la țintă;
- montarea trebuie să permită un grad ridicat de libertate pentru mișcarea oglinzii și să fie echilibrate pentru vânturi și alte perturbări asigurând os stabilitate mecanică suficientă - vibrațiile mari ar trebui excluse;

Pentru poziționarea oglinzii a fost selectata o construcție electromecanică ce permite manipularea oglinzii pe doua axe de rotație cum e prezentat în Fig. A3.2.

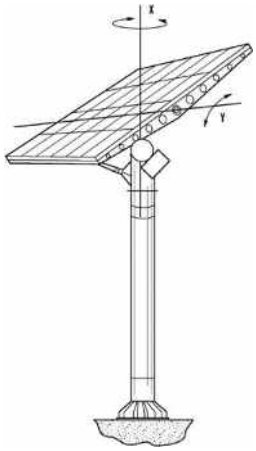


Fig. A3.2. Sistem electromecanic de poziționare a oglinzii reflectoare

Cerințe electrice față de sistem

Reieșind din construcția mecanica și cerințele generale de sistem cerințele electrice sunt prezentate după cum urmează:

- Sistemul electric trebuie alimentat de la sursa de 220 V 60 Hz AC.
- Sistemul electronic trebuie sa fie dotat cu senzori de măsurare precisă a două unghiuri care definesc configurația curentă a oglinzii.

- Sistemul electronic trebuie să furnizeze o alimentare electrica de 24V până la 3A DC suficient pentru a conduce două motoare.
- Motoarele de poziționare a oglinzilor trebuie să furnizeze suficientă putere (cuplu) pentru a controla mecanismul de rotație pe cele doua axe.
- Motoarele de poziționare a oglinzilor trebuie să poată efectua deplasări unghiulare relativ mici, astfel încât oglinda să poată fi reglată cu precizie relativ la poziția soarelui.
- În scop de protecție sistemul electronic trebuie sa fie dotat cu limitatoare a unghiurilor conectate electric direct la motoare implementând un concept de blocare de siguranță în caz de depașire a unghiului admisibil, astfel încât oglinzile să nu fie deteriorate.
- Sistemul trebuie sa fie dotat cu senzor de măsurare a vântului.
- Sistemul trebuie sa aibă protecție la curenți mari conform specificațiilor motoarelor
- Opțional poate fi inclusă o interfața cu utilizatorul pentru configurarea locală a sistemului realizata prin componente de interacțiune cum ar fi buton, led, encoder și semnalizator sonor.

Conform cerințelor electrice expuse mai sus a fost realizata o unitate electrica de control - ECU prezentată în Fig. A3.3.

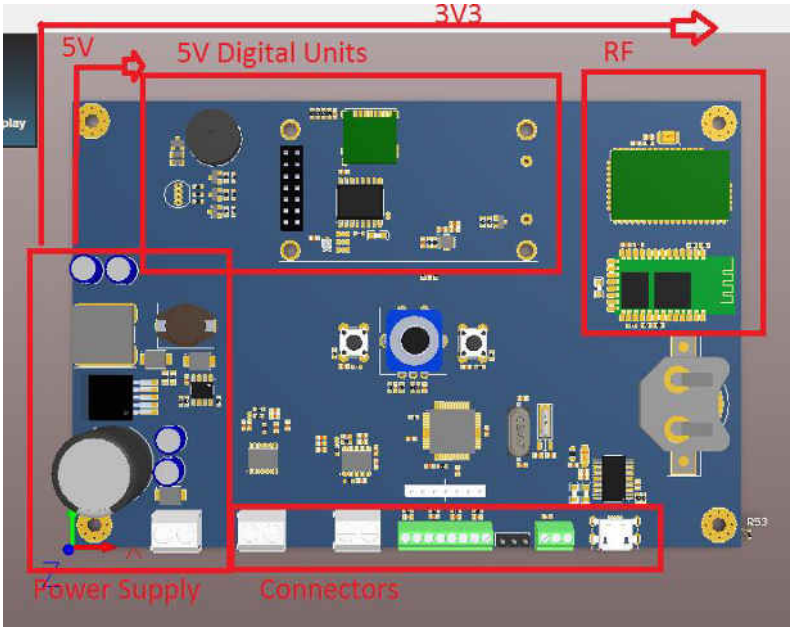


Fig. A3.3. Unitatea electrică de control al sistemului

Sistemul de reflecție a luminilor în ansamblu instalat pe clădirea din Pavilionul Republicii Moldova la expoziția din Milano 2015 se poate vedea în Fig. A3.4.

Fig. A3.4. Clădirea cu sistemul de reflecție a luminii instalat

Anexa 4. Configurații ale proiectelor în format JSON

Cofiguratia JSON pentru dispozitivul IoT

```
{
  "Description": "Application Demo for IoT
Environment sensor Device  control",
  "git": "https://github.com/ML-ES-
Platform/env_sns_dev_demo.git",
  "Path": "ASW/env_sns_dev_demo/",
  "Components": {
    "MCU_IO": {
      "Layer": "MCU",
      "Groups": {
        "DI_PIN": {
          "Channels": {
            "D1": [],
            "D2": [],
            "D3": []
          }
        },
        "DO_PIN": {
          "Channels": {
            "D11": [],
            "D12": [],
            "D13": []
          }
        },

        "PORT_PIN": {
          "Channels": {
          }
        },

        "ADC_PIN": {
          "Channels": {
            "A3": [],
            "A4": []
          }
        },
        "PWM_PIN": {
          "Channels": {

          }
        },
        "ONE_WIRE_IF": {
          "Channels": {
            "if_1w": []
          }
        },
        "I2C_IF": {
          "Channels": {

          }
        },
        "SPI_IF": {
          "Channels": {
            "spi": []
          }
        },
        "USART_IF": {
          "Channels": {

          }
        }
       }
    },
    "mcal_din": {
      "git": "https://github.com/ML-ES-
Platform/mcal_din.git",
      "Path": "MCAL/mcal_din/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DI_1": ["D1"],
            "DI_2": ["D2"],
            "DI_3": ["D3"]
          }
        }
      }
    },
    "mcal_dout": {
      "git": "https://github.com/ML-ES-
Platform/mcal_dout.git",
      "Path": "MCAL/mcal_dout/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DO_1": ["D11"],
            "DO_2": ["D12"],
            "DO_3": ["D13"]
          }
        }
      }
    },
    "mcal_adc": {
      "git": "https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_adc": {
          "Channels": {
            "ADC_1": ["A3"],
            "ADC_2": ["A4"]
          }
        }
      }
    },
    "mcal_pwm": {
      "git": "https://github.com/ML-ES-
Platform/mcal_pwm.git",
      "Path": "MCAL/mcal_pwm/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_pwm": {
          "Channels": {

          }
        }
      }
    },
```

```
    "mcal_i2c": {
      "git": "https://github.com/ML-ES-
Platform/mcal_i2c.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_i2c": {
          "Channels": {
          }
        }
      }
    },
    "mcal_1w": {
      "git": "https://github.com/ML-ES-
Platform/mcal_1w.git",
      "Path": "MCAL/mcal_1w/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_1w": {
          "Channels": {
            "MCAL_1W": ["if_1w"]
          }
        }
      }
    },
    "mcal_spi": {
      "git": "https://github.com/ML-ES-
Platform/mcal_spi.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_spi": {
          "Channels": {
            "MCAL_SPI": ["spi"]
          }
        }
      }
    },
    "mcal_usart": {
      "git": "https://github.com/ML-ES-
Platform/mcal_usart.git",
      "Path": "MCAL/mcal_usart/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_usart": {
          "Channels": {
          }
        }
      }
    },
    "dd_contact": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_contact": {
          "Channels": {
            "CNT_1": ["DI_3"]
          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_switch": {
      "git": "https://github.com/ML-ES-
Platform/dd_switch.git",
      "Path": "ESW/dd_switch/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_ls_switch": {
          "Channels": {
            "LSSW_1": ["DO_1"],
            "LSSW_2": ["DO_2"],
            "LSSW_3": ["DO_3"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_anacomp": {
      "git": "https://github.com/ML-ES-
Platform/dd_anacomp.git",
      "Path": "ESW/dd_anacomp/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_anacomp": {
          "Channels": {
            "AC_1": ["DI_1"],
            "AC_2": ["DI_2"]
          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_ldr": {
      "git": "https://github.com/ML-ES-
Platform/dd_ldr.git",
      "Path": "ESW/dd_ldr/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_anacomp": {
          "Channels": {
            "ldr_light": ["ADC_1"]
          },
          "Pull": "ADC_ReadChannel",
          "Dependency": "mcal_adc"
        }
      }
    },
    "dd_mq135": {
      "git": "https://github.com/ML-ES-
Platform/dd_mq135.git",
      "Path": "ESW/dd_mq135/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_mq135": {
          "Channels": {
            "mq135_co2": ["ADC_2"]
          },
          "Pull": "ADC_ReadChannel",
          "Dependency": "mcal_adc"
        }
      }
```

```
    },

    "dd_mrf24": {
      "git": "https://github.com/ML-ES-
Platform/dd_mrf24j40ma.git",
      "Path": "ESW/dd_mrf24j40ma/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_tcp_ip": {
          "Channels": {
            "mrf_zigbee": ["MCAL_SPI"]
          },
          "Dependency": "mcal_spi",
          "Pull": "SPI_TransferByte"
        }
      }
    },


    "dd_dht_xx": {
      "git": "https://github.com/ML-ES-
Platform/dd_dht_xx.git",
      "Path": "ESW/dd_dht_xx/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_humidity": {
          "Channels": {
            "dht_hum": ["MCAL_1W"]
          },
          "Pull": "OW_ReadChannel",
          "Dependency": "mcal_1w"
        },
        "dd_temp": {
          "Channels": {
            "dht_temp": ["MCAL_1W"]
          },
          "Pull": "OW_ReadChannel",
          "Dependency": "mcal_1w"
        }
      }
    },
    "vd_com_zigbee": {
      "git": "https://github.com/ML-ES-
Platform/vd_com_zigbee.git",
      "Path": "ESW/vd_com_zigbee/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_tcp_ip": {
          "Channels": {
            "ZIGBEE": ["mrf_zigbee"]
          },
          "Dependency": "dd_mrf24j40ma",
          "Pull": "MRF_Transfer"
        }
      }
    },

    "vd_sns_temperature": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_temperature.git",
      "Path": "ESW/vd_sns_temperature/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_humidity": {
          "Channels": {
            "TEMP": ["dht_temp"]
          },
          "Dependency": "dd_dht_xx",
          "Pull": "DHT_ReadTemperatuire"
        }
      }
    },
    "vd_sns_humidity": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_humidity.git",
      "Path": "ESW/vd_sns_humidity/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_humidity": {
          "Channels": {
            "HUM": ["dht_hum"]
          },
          "Dependency": "dd_dht_xx",
          "Pull": "DHT_ReadHumidity"
        }
      }
    },
    "vd_sns_co2": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_co2.git",
      "Path": "ESW/vd_sns_co2/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_co2": {
          "Channels": {
            "CO2": ["mq135_co2"]
          },
          "Dependency": "dd_mq135",
          "Pull": "mq135_ReadCo2"
        }
      }
    },

    "vd_sns_light": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_light.git",
      "Path": "ESW/vd_sns_light/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_light": {
          "Channels": {
            "LIGHT": ["ldr_light"]
          },
          "Dependency": "dd_ldr",
          "Pull": "ldr_ReadLight"
        }
      }
    },

    "vd_sns_dig_mic": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_dig_mic.git",
      "Path": "ESW/vd_sns_dig_mic/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_dig_mic": {
          "Channels": {
            "DMIC": ["AC_1"]
```

```
          },
          "Dependency": "dd_anacomp",
          "Pull": "AC_ReadChannel"
        }
      }
    },
    "vd_sns_pir": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_pir.git",
      "Path": "ESW/vd_sns_pir/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_pir": {
          "Channels": {
            "PIR": ["AC_2"]
          },
          "Dependency": "dd_anacomp",
          "Pull": "AC_ReadChannel"
        }
      }
    },
    "vd_ui_led_rgb": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_led_rgb.git",
      "Path": "ESW/vd_ui_led_rgb/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "led_r": {
          "Channels": {
            "LED_RGB":
["LSSW_1","LSSW_2","LSSW_3"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },

    "vd_ui_button": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_button.git",
      "Path": "ESW/vd_ui_button/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_button": {
          "Channels": {
            "BUTTON": ["CNT_1"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "os_time_trig": {
      "git": "https://github.com/ML-ES-
Platform/os_time_trig.git",
      "Path": "BSW/os_time_trig/",
      "Layer": "OS",
      "Domain": "OS"
    },
    "rte_com": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "zigbee": {
          "Channels": {
            "COM_ZIGBEE": ["ZIGBEE"]
          }
        }
      }
    },
     "rte_sns": {
      "git": "https://github.com/ML-ES-
Platform/rte_sns.git",
      "Path": "BSW/rte_sns/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "sns_dig_mic": {
          "Channels": {
            "SNS_DMIC": ["DMIC"]
          }
        },
        "sns_pir": {
          "Channels": {
            "SNS_PIR": ["PIR"]
          }
        },
        "sns_humidity": {
          "Channels": {
            "SNS_HUM": ["HUM"]
          }
        },
        "sns_temperature": {
          "Channels": {
            "SNS_TEMP": ["TEMP"]
          }
        },
        "sns_co2": {
          "Channels": {
            "SNS_CO2": ["CO2"]
          }
        },
        "sns_light": {
          "Channels": {
            "SNS_LIGHT": ["LIGHT"]
          }
        }
      }
    },
    "rte_ui": {
      "git": "https://github.com/ML-ES-
Platform/rte_ui.git",
      "Path": "BSW/rte_ui/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "led_rgb": {
          "Channels": {
            "UI_LED_RGB": ["LED_RGB"]
          }
        },
        "button": {
          "Channels": {
            "UI_BUTTON": ["BUTTON"]
          }
        }
      }
    },
```

```
    "comp_diagnostic": {
      "TypeName": "Diagnostic",
      "git": "https://github.com/ML-ES-
Platform/diagnostic.git",
      "Path": "ASW/diagnostic/",
      "Layer": "APP_Sensor_Dev",
      "Domain": "APP_Sensor_Dev",
      "Groups": {
        "diagnostic": {
          "Channels": {

            "temp_diag" : ["SNS_TEMP"  ],
            "hum_diag"  : ["SNS_HUM"   ],
            "co2_diag"  : ["SNS_CO2"   ],
            "light_diag": ["SNS_LIGHT" ],
            "dmic_diag" : ["SNS_DMIC"  ],
            "pir_diag"  : ["SNS_PIR"   ],
            "led_alert" : ["UI_LED_RGB"]
          }
        }
      }
    },
    "comp_streaming": {
      "TypeName": "dev_ctrl",
      "git": "https://github.com/ML-ES-
Platform/comp_streaming.git",
      "Path": "ASW/comp_streaming/",
      "Layer": "APP_Sensor_Dev",
      "Domain": "APP_Sensor_Dev",
      "Groups": {
        "streaming": {
          "Channels": {
            "Streaming": ["COM_ZIGBEE"]
          }
        }
      }
    },
    "comp_Ctrl": {
      "TypeName": "dev_ctrl",
      "git": "https://github.com/ML-ES-
Platform/dev_ctrl.git",
      "Path": "ASW/dev_ctrl/",
      "Layer": "APP_Sensor_Dev",
      "Domain": "APP_Sensor_Dev",
      "Groups": {
        "ui_intf": {
          "Channels": {
            "ctrl_ui":
["UI_BUTTON","led_alert"]
          }
        },
        "sensor_diag": {
          "Channels": {
            "sns_diag":
["temp_diag","hum_diag","co2_diag","light_diag
","dmic_diag","pir_diag"]
          }
        },
        "ctrl_com": {
          "Channels": {
            "ctrl_com": ["Streaming"]
          }
        }
      }
    }
  }
}
```

Cofiguratia JSON pentru Serverul IoT

```
{
  "Description": "Application Demo for IoT
Environment sensor Server control",
  "ProjectName": "env_sns_srv",

  "git": "https://github.com/ML-ES-
Platform/env_sns_dev_demo.git",
  "Path": "ASW/env_sns_dev_demo/",
  "Components": {
    "MCU_IO": {
      "Layer": "MCU",
      "Domain": "ECU",
      "Groups": {
        "DI_PIN": {
          "Channels": {
            "D3": []
          }
        },
        "DO_PIN": {
          "Channels": {
            "D11": [],
            "D12": [],
            "D13": []
          }
        },

        "PORT_PIN": {
          "Channels": {
            "P1": []
          }
        },

        "MCU_ADC_PIN": {
          "Channels": {

          }
        },
        "PWM_PIN": {
          "Channels": {

          }
        },
        "ONE_WIRE_IF": {
          "Channels": {
          }
        },
        "I2C_IF": {
          "Channels": {

          }
        },
        "SPI_IF": {
          "Channels": {
            "spi": []
          }
        },
        "USART_IF": {
          "Channels": {

          }
        }

      }
    },
    "mcal_din": {
      "git": "https://github.com/ML-ES-
Platform/mcal_din.git",
      "Path": "MCAL/mcal_din/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DI_3": ["D3"]
          }
        }
      }
    },
    "mcal_dout": {
      "git": "https://github.com/ML-ES-
Platform/mcal_dout.git",
      "Path": "MCAL/mcal_dout/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DO_1": ["D11"],
            "DO_2": ["D12"],
            "DO_3": ["D13"]
          }
        }
      }
    },
    "mcal_port": {
      "git": "https://github.com/ML-ES-
Platform/mcal_port.git",
      "Path": "MCAL/mcal_port/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "PORT_1": ["P1"]
          }
        }
      }
    },
    "mcal_adc": {
      "git": "https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_adc": {
          "Channels": {
          }
        }
      }
    },
    "mcal_pwm": {
      "git": "https://github.com/ML-ES-
Platform/mcal_pwm.git",
      "Path": "MCAL/mcal_pwm/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_pwm": {
          "Channels": {
```

```
          }
        }
      }
    },
    "mcal_i2c": {
      "git": "https://github.com/ML-ES-
Platform/mcal_i2c.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_i2c": {
          "Channels": {
          }
        }
      }
    },
    "mcal_1w": {
      "git": "https://github.com/ML-ES-
Platform/mcal_1w.git",
      "Path": "MCAL/mcal_1w/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_1w": {
          "Channels": {
          }
        }
      }
    },
    "mcal_spi": {
      "git": "https://github.com/ML-ES-
Platform/mcal_spi.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_spi": {
          "Channels": {
            "MCAL_SPI": ["spi"]
          }
        }
      }
    },
    "mcal_usart": {
      "git": "https://github.com/ML-ES-
Platform/mcal_usart.git",
      "Path": "MCAL/mcal_usart/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_usart": {
          "Channels": {
          }
        }
      }
    },
    "dd_contact": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_contact": {
          "Channels": {
            "CNT_1": ["DI_3"]
          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_switch": {
      "git": "https://github.com/ML-ES-
Platform/dd_switch.git",
      "Path": "ESW/dd_switch/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_ls_switch": {
          "Channels": {
            "LSSW_1": ["DO_1"],
            "LSSW_2": ["DO_2"],
            "LSSW_3": ["DO_3"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_gr_lcd": {
      "git": "https://github.com/ML-ES-
Platform/dd_gr_lcd.git",
      "Path": "ESW/dd_gr_lcd/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_ls_switch": {
          "Channels": {
            "GR_LCD": ["PORT_1"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dio"
        }
      }
    },

    "dd_mrf24": {
      "git": "https://github.com/ML-ES-
Platform/dd_mrf24j40ma.git",
      "Path": "ESW/dd_mrf24j40ma/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "dd_mrf24": {
          "Channels": {
            "mrf_zigbee": ["MCAL_SPI"]
          },
          "Dependency": "mcal_spi",
          "Pull": "SPI_TransferByte"
        }
      }
    },
    "dd_esp32wifi": {
      "git": "https://github.com/ML-ES-
Platform/dd_esp32wifi.git",
      "Path": "ESW/dd_mrf24j40ma/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "dd_esp32wifi": {
          "Channels": {
            "esp32_wifi": ["MCAL_SPI"]
          },
```

```
          "Dependency": "mcal_spi",
          "Pull": "SPI_TransferByte"
        }
      }
    },

    "vd_com_zigbee": {
      "git": "https://github.com/ML-ES-
Platform/vd_com_zigbee.git",
      "Path": "ESW/vd_com_zigbee/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_zigbee": {
          "Channels": {
            "ZIGBEE": ["mrf_zigbee"]
          },
          "Dependency": "dd_mrf24j40ma",
          "Pull": "MRF_Transfer"
        }
      }
    },

    "vd_com_tcp_ip": {
      "git": "https://github.com/ML-ES-
Platform/vd_com_tcp_ip.git",
      "Path": "ESW/vd_com_tcp_ip/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_tcp_ip": {
          "Channels": {
            "TCP_IP": ["esp32_wifi"]
          },
          "Dependency": "mcal_spi",
          "Pull": "SPI_TransferByte"
        }
      }
    },


    "vd_ui_led_rgb": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_led_rgb.git",
      "Path": "ESW/vd_ui_led_rgb/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "led_r": {
          "Channels": {
            "LED_RGB":
["LSSW_1","LSSW_2","LSSW_3"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },

    "vd_ui_gr_display": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_gr_display.git",
      "Path": "ESW/vd_ui_gr_display/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "led_r": {
          "Channels": {
            "DISPLAY": ["GR_LCD"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },

    "vd_ui_button": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_button.git",
      "Path": "ESW/vd_ui_button/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_button": {
          "Channels": {
            "BUTTON": ["CNT_1"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },

    "os_time_trig": {
      "git": "https://github.com/ML-ES-
Platform/os_time_trig.git",
      "Path": "BSW/os_time_trig/",
      "Layer": "OS",
      "Domain": "OS"
    },
    "rte_com": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {

        "tcp_ip": {
          "Channels": {
            "COM_TCP_IP": ["TCP_IP"]
          }
        }

      }
    },
     "rte_sns": {
      "git": "https://github.com/ML-ES-
Platform/rte_sns.git",
      "Path": "BSW/rte_sns/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {

        "zigbee": {
          "Channels": {
            "COM_ZIGBEE": ["ZIGBEE"]
          }
        },
        "sns_dig_mic": {
          "Channels": {
            "SNS_DMIC": ["COM_ZIGBEE"]
          }
        },
        "sns_pir": {
          "Channels": {
```

```
            "SNS_PIR": ["COM_ZIGBEE"]
          }
        },
        "sns_humidity": {
          "Channels": {
            "SNS_HUM": ["COM_ZIGBEE"]
          }
        },
        "sns_temperature": {
          "Channels": {
            "SNS_TEMP": ["COM_ZIGBEE"]
          }
        },
        "sns_co2": {
          "Channels": {
            "SNS_CO2": ["COM_ZIGBEE"]
          }
        },
        "sns_light": {
          "Channels": {
            "SNS_LIGHT": ["COM_ZIGBEE"]
          }
        }
      }
    },
    "rte_ui": {
      "git": "https://github.com/ML-ES-
Platform/rte_ui.git",
      "Path": "BSW/rte_ui/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "led_rgb": {
          "Channels": {
            "UI_LED_RGB": ["LED_RGB"]
          }
        },
        "button": {
          "Channels": {
            "UI_BUTTON": ["BUTTON"]
          }
        },
        "display": {
          "Channels": {
            "UI_DISPLAY": ["DISPLAY"]
          }
        }
      }
    },

    "comp_real_time": {
      "TypeName": "Diagnostic",
      "git": "https://github.com/ML-ES-
Platform/diagnostic.git",
      "Path": "ASW/diagnostic/",
      "Layer": "APP_Sensor_SRVR",
      "Domain": "APP_Sensor_SRVR",
      "Groups": {
        "real_time_sns": {
          "Channels": {
            "temp_rt": ["SNS_TEMP"],
            "hum_rt":  ["SNS_HUM"],
            "co2_rt":  ["SNS_CO2"],
            "light_rt":["SNS_LIGHT"],
            "dmic_rt": ["SNS_DMIC"],
            "pir_rt":  ["SNS_PIR"],
            "show_rt": ["UI_DISPLAY"]
          }
```

```
        }

      }
    },
    "comp_map_int": {
      "TypeName": "Diagnostic",
      "git": "https://github.com/ML-ES-
Platform/diagnostic.git",
      "Path": "ASW/diagnostic/",
      "Layer": "APP_Sensor_SRVR",
      "Domain": "APP_Sensor_SRVR",
      "Groups": {
        "comp_map_int": {
          "Channels": {

            "temp_val": ["temp_rt"],
            "hum_val":  ["hum_rt"],
            "co2_val":  ["co2_rt"],
            "light_val":["light_rt"],
            "dmic_val": ["dmic_rt"],
            "pir_val":  ["pir_rt"],
            "show_map": ["UI_DISPLAY"]
          }
        }
      }
    },
    "comp_streaming": {
      "TypeName": "dev_ctrl",
      "git": "https://github.com/ML-ES-
Platform/comp_streaming.git",
      "Path": "ASW/comp_streaming/",
      "Layer": "APP_Sensor_SRVR",
      "Domain": "APP_Sensor_SRVR",
      "Groups": {
        "streaming": {
          "Channels": {
            "Streaming": ["COM_TCP_IP"]
          }
        }
      }
    },
    "comp_Ctrl": {
      "TypeName": "dev_ctrl",
      "git": "https://github.com/ML-ES-
Platform/dev_ctrl.git",
      "Path": "ASW/dev_ctrl/",
      "Layer": "APP_Sensor_SRVR",
      "Domain": "APP_Sensor_SRVR",
      "Groups": {
        "ui_button": {
          "Channels": {
            "ctrl_ui": ["UI_BUTTON"]
          }
        },
        "ui_led_rgb": {
          "Channels": {
            "ctrl_ui": ["UI_LED_RGB"]
          }
        },
        "ui_display": {
          "Channels": {
            "ctrl_ui": ["UI_DISPLAY"]
          }
        },
        "ctrl_graph": {
          "Channels": {
            "ctrl_graph": ["ctrl_ui"]
          }
```

```
        },
        "ctrl_map": {
          "Channels": {
            "ctrl_graph": ["show_map"]
          }
        },
        "ctrl_rt": {
          "Channels": {
            "ctrl_graph": ["show_rt"]
          }
        },
        "ctrl_com": {
          "Channels": {
            "ctrl_com": ["Streaming"]
```

```
          }
        }
      }
    }
  }
}
```

Cofiguratia JSON pentru sistem de control 3 DOF

```
{
  "Description": "Application Demo for 3-DOF
Robotic arm  control",
  "ProjectName": "arm_3dof_demo",
  "git": "https://github.com/ML-ES-
Platform/arm_3dof_demo.git",
  "Path": "ASW/arm_3dof_demo/",
  "Components": {
    "PCA9685_DEV": {
      "Domain": "ECU",
      "Layer": "ECU",

      "Groups": {
        "PCA9685_PIN": {
          "Discipline": "HW",
          "Channels": {
            "PCA9685_PIN_1": [],
            "PCA9685_PIN_2": [],
            "PCA9685_PIN_3": []
          }
        },
        "PCA9685_DEV": {
          "Discipline": "HW",
          "Channels": {
            "PCA9685_DEV":
["PCA9685_PIN_1","PCA9685_PIN_2","PCA9685_PIN_3
"]
          }
        }
      }
    },
    "MCU_IO": {
      "Layer": "MCU",
      "Domain": "ECU",
      "Groups": {
        "DI_PIN": {
          "Channels": {
          }
        },
        "DO_PIN": {
          "Channels": {
          }
        },

        "PORT_PIN": {
          "Channels": {
          }
```

```
        },

        "MCU_ADC_PIN": {
          "Discipline":"HW",
          "Channels": {
            "A3": [],
            "A4": [],
            "A5": []
          }
        },
        "PWM_PIN": {
          "Channels": {
          }
        },
        "ONE_WIRE_IF": {
          "Channels": {
          }
        },
        "I2C_IF": {
          "Channels": {
          }
        },
        "SPI_IF": {
          "Channels": {
          }
        },
        "USART_IF": {
          "Channels": {
          }
        }

      }
    },
    "mcal_adc": {
      "git": "https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_adc": {
          "Channels": {
            "ADC_1": ["A3"],
            "ADC_2": ["A4"],
            "ADC_3": ["A5"]
          }
        }
```

```
      }
    },
    "mcal_pwm": {
      "git": "https://github.com/ML-ES-
Platform/mcal_pwm.git",
      "Path": "MCAL/mcal_pwm/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_pwm": {
          "Channels": {
          }
        }
      }
    },

    "mcal_i2c": {
      "git": "https://github.com/ML-ES-
Platform/mcal_i2c.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_i2c": {
          "Channels": {
            "MCAL_I2C": ["PCA9685_DEV"]
          }
        }
      }
    },

    "dd_pot": {
      "git": "https://github.com/ML-ES-
Platform/dd_potentiometer.git",
      "Path": "ESW/dd_pot/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_pot": {
          "Channels": {
            "POT_1": ["ADC_1"],
            "POT_2": ["ADC_2"],
            "POT_3": ["ADC_3"]
          },
          "Dependency": "mcal_adc",
          "Pull": "ADC_ReadChannel"
        }
      }
    },


    "dd_pca9685": {
      "git": "https://github.com/ML-ES-
Platform/dd_pca9685.git",
      "Path": "ESW/dd_pca9685/",
      "Layer": "ECAL",
      "Domain": "ACT",
      "Groups": {
        "dd_pca9685": {
          "Channels": {
            "PCA9685_PWM_1": ["MCAL_I2C"],
            "PCA9685_PWM_2": ["MCAL_I2C"],
            "PCA9685_PWM_3": ["MCAL_I2C"]
          },
          "Defines": {
            "PCA9685_PWM_FREQ": 50
          }
        }
```

```
      }
    },
    "dd_servo": {
      "Name": "dd_servo",
      "git": "https://github.com/ML-ES-
Platform/dd_servo.git",
      "Path": "ESW/dd_servo/",
      "Layer": "ECAL",
      "Domain": "ACT",
      "Groups": {
        "dd_servo": {
          "Name": "dd_servo",
          "Channels": {
            "DD_SERVO_1": ["PCA9685_PWM_1"],
            "DD_SERVO_2": ["PCA9685_PWM_2"],
            "DD_SERVO_3": ["PCA9685_PWM_3"]
          },
          "Dependency": "dd_pca9685",
          "Push": "PCA9685_WriteChannel"
        }
      }
    },
    "vd_sns_angle": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_angle.git",
      "Path": "ESW/vd_sns_angle/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_angle": {
          "Channels": {
            "ANGSNS_1": ["POT_1"],
            "ANGSNS_2": ["POT_2"],
            "ANGSNS_3": ["POT_3"]
          },
          "Pull": "POT_GetPosition"
        }
      }
    },
    "vd_act_servo": {
      "Name": "vd_act_servo",
      "git": "https://github.com/ML-ES-
Platform/vd_act_servo.git",
      "Path": "ESW/vd_act_servo/",
      "Layer": "SRV",
      "Domain": "ACT",
      "Groups": {
        "vd_act_servo": {
          "Name": "vd_act_servo",
          "Channels": {
            "VD_ACT_SERVO_1": ["DD_SERVO_1"],
            "VD_ACT_SERVO_2": ["DD_SERVO_2"],
            "VD_ACT_SERVO_3": ["DD_SERVO_3"]
          },
          "Runnable": {
            "Task":
"VD_ACT_SERVOChannelRunnable",
            "Recurrence": "SERVO_TASK_REC",
            "Timeout": "SERVO_TASK_OFFSET"
          },
          "Defines": {
            "SECOND_MS": 1000.0,
            "SERVO_TASK_REC": 10,
            "SERVO_TASK_OFFSET": 100
          },
          "Dependency": "dd_servo",
          "Push": "DDSERVO_SetAngle"
        }
```

```
      }
    },
    "os_time_trig": {
      "git": "https://github.com/ML-ES-
Platform/os_time_trig.git",
      "Path": "BSW/os_time_trig/"
    },
    "rte_com": {
      "git": "https://github.com/ML-ES-
Platform/rte_com.git",
      "Path": "BSW/rte_com/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Ignore": "true",
      "Groups": {
        "i2c": {
          "Channels": {

          }
        }
      }
    },
    "rte_act": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "sns_ang_fb": {
          "Channels": {
            "ACT_SRV_1":["VD_ACT_SERVO_1"],
            "ACT_SRV_2":["VD_ACT_SERVO_2"],
            "ACT_SRV_3":["VD_ACT_SERVO_3"]
          }
        }
      }
    },
     "rte_sns": {
      "git": "https://github.com/ML-ES-
Platform/rte_sns.git",
      "Path": "BSW/rte_sns/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "sns_ang_fb": {
          "Channels": {
            "SNS_ANG_1":["ANGSNS_1"],
            "SNS_ANG_2":["ANGSNS_2"],
            "SNS_ANG_3":["ANGSNS_3"]
          }
        },
        "sns_pir": {
          "Channels": {
          }
        },
        "sns_humidity": {
          "Channels": {
          }
        },
        "sns_temperature": {
          "Channels": {
          }
        },
        "sns_co2": {
          "Channels": {
          }
        },
        "sns_light": {
          "Channels": {
          }
        }
      }
    },
    "rte_ui": {
      "git": "https://github.com/ML-ES-
Platform/rte_ui.git",
      "Path": "BSW/rte_ui/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Ignore": "true",
      "Groups": {
        "led_rgb": {
          "Channels": {
          }
        },
        "button": {
          "Channels": {
          }
        }
      }
    },

    "comp_arm_3dof": {
      "git": "https://github.com/ML-ES-
Platform/arm_3dof.git",
      "Path": "ASW/arm_3dof/",
      "Name": "arm_3dof",
      "Groups": {
        "ARM_3DOF": {
          "Channels": {
            "ARM_3DOF_1": [ "POS_XYZ_SET",
"POS_XYX_FB"]
          }
        },
        "ARM_3DOF_Servo": {
          "Parent": "ARM_3DOF_1",
          "Channels": {
            "ACT_BASE": ["ACT_SRV_1"],
            "ACT_L1"  : ["ACT_SRV_2"],
            "ACT_L2"  : ["ACT_SRV_3"]
          },
          "Channels_x": {
            "ACT_ROLL" : [ ],
            "ACT_PITCH": [ ],
            "ACT_CLAW" : [ ]
          },
          "Push": "VD_ACT_SERVOAngleSet"
        },
        "ARM_3DOF_Segment": {
          "Name": "arm_3dof_Segment",
          "Parent": "ARM_3DOF_1",
          "Channels": {
            "SEG_BASE": [ "INV_CNM",
"DIR_CNM"],
            "SEG_L1"  : [ "INV_CNM",
"DIR_CNM"],
            "SEG_L2"  : [ "INV_CNM", "DIR_CNM"]
          }
        },

        "ARM_3DOF_PosSet": {
          "Parent": "ARM_3DOF_1",
          "Channels": {
            "POS_XYZ_SET": ["INV_CNM"],
```

```
            "INV_CNM": ["ACT_BASE", "ACT_L1",
"ACT_L2"]
          },
          "Pull": "REV_CNM_GetPos"
        },

        "ARM_3DOF_AnglesFB": {
          "Parent": "ARM_3DOF_1",
          "Channels": {
            "ANG_FB_BASE": ["SNS_ANG_1"],
            "ANG_FB_L1"  : ["SNS_ANG_2"],
            "ANG_FB_L2"  : ["SNS_ANG_3"]
          },
          "Pull": "ANGSNS_GetAngle"
        },
        "ARM_3DOF_PosFB": {
          "Parent": "ARM_3DOF_1",
          "Channels": {
            "POS_XYX_FB": ["DIR_CNM"],
            "DIR_CNM": ["ANG_FB_BASE",
"ANG_FB_L1", "ANG_FB_L2"]
          },
          "Pull": "REV_CNM_GetPos"
        },
        "ARM_3DOF_Coord": {
          "Ignore": "true",
          "Channels": {
            "COORD_X": [ ],
            "COORD_Y": [ ],
            "COORD_Z": [ ],
            "COORD_XYZ": [ ]
          }
        }
      },
      "Features": {
        "ARM_control": {
          "Channels": [
            "ACT_BASE",
            "ACT_L1",
            "ACT_L2",
            "ANG_FB_BASE",
            "ANG_FB_L1",
            "ANG_FB_L2"
          ]
        }
```

```
      }
    },
    "arm_3dof_demo": {
      "git": "https://github.com/ML-ES-
Platform/arm_3dof.git",
      "Path": "ASW/arm_3dof/",
      "Ignore": "true",
      "Groups": {
        "ARMS_3DOF_demo": {
          "Discipline":"null",
          "Channels": {
            "ARM_X": ["COORD_X"],
            "ARM_Y": ["COORD_Y"],
            "ARM_Z": ["COORD_Z"]
          },
            "Channels_x": {
            "ARM_CLAW_angle": [ ],
            "ARM_CLAW_open": [ ]
          },
          "Dependency": "arm_3dof",
          "Push": "SetCoord"
        },
        "ARMS_3DOF": {
          "Discipline":"null",
          "Channels": {
            "ARM1": [ ]
          },
          "Dependency": "arm_3dof",
          "Push": "SetXYZ"
        }
      }
    }
  },
  "FileName": "arm_3dof_cfg_tst.json",
  "HomeDir": "C:/Users/Admin/OneDrive -
Technical University of
Moldova/MicroLabOS_WS/ES_Platform/src"
}
```

Cofiguratia JSON pentru sistem de control refelectie lumina

```
{
  "Description": "Application Demo for
Heliostat control",
  "ProjectName": "heliostat",

  "git": "https://github.com/ML-ES-
Platform/heliostat.git",
  "Path": "ASW/heliostat/",
  "Components": {
    "MCU_IO": {
      "Layer": "MCU",
      "Groups": {
        "DI_PIN": {
          "Channels": {
            "D1": [],
            "D2": [],
            "D3": [],
            "D4": [],
            "D5": []
          }
        },
        "DO_PIN": {
          "Channels": {
            "D11": [],
            "D12": [],
            "D13": [],
            "D14": [],
            "D15": [],
            "D16": []
          }
        },
        "ADC_PIN": {
          "Channels": {

          }
        },
        "PWM_PIN": {
          "Channels": {
            "PWM1": [],
            "PWM2": []
          }
        },
        "ONE_WIRE_IF": {
          "Channels": {
          }
        },
        "I2C_IF": {
          "Channels": {
            "i2c": []
          }
        },
        "SPI_IF": {
          "Channels": {
            "spi": []
          }
        },
        "USART_IF": {
          "Channels": {
            "usart": []
          }
        }
      }
    },
```

```
    "mcal_din": {
      "git": "https://github.com/ML-ES-
Platform/mcal_din.git",
      "Path": "MCAL/mcal_din/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DI_1": ["D1"],
            "DI_2": ["D2"],
            "DI_3": ["D3"],
            "DI_4": ["D4"],
            "DI_5": ["D5"]
          }
        }
      }
    },
    "mcal_dout": {
      "git": "https://github.com/ML-ES-
Platform/mcal_dout.git",
      "Path": "MCAL/mcal_dout/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DO_1": ["D11"],
            "DO_2": ["D12"],
            "DO_3": ["D13"],
            "DO_4": ["D14"],
            "DO_5": ["D15"],
            "DO_6": ["D16"]
          }
        }
      }
    },
    "mcal_adc": {
      "git": "https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_adc": {
          "Channels": {}
        }
      }
    },
    "mcal_pwm": {
      "git": "https://github.com/ML-ES-
Platform/mcal_pwm.git",
      "Path": "MCAL/mcal_pwm/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_pwm": {
          "Channels": {
            "PWM_1": ["PWM1"],
            "PWM_2": ["PWM2"]
          }
        }
      }
```

```
    },
    "mcal_i2c": {
      "git": "https://github.com/ML-ES-
Platform/mcal_i2c.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_i2c": {
          "Channels": {
            "MCAL_I2C": ["i2c"]
          }
        }
      }
    },
    "mcal_spi": {
      "git": "https://github.com/ML-ES-
Platform/mcal_spi.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_spi": {
          "Channels": {
            "mcal_spi":["spi"]
          }
        }
      }
    },
    "mcal_usart": {
      "git": "https://github.com/ML-ES-
Platform/mcal_usart.git",
      "Path": "MCAL/mcal_usart/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_usart": {
          "Channels": {
            "mcal_usart": ["usart"]
          }
        }
      }
    },
    "dd_cnt_sns": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_contact": {
          "Channels": {
            "CNT_4": ["DI_4"],
            "CNT_5": ["DI_5"]
          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_cnt_ui": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_contact": {
          "Channels": {
            "CNT_1": ["DI_1"],
            "CNT_2": ["DI_2"],
            "CNT_3": ["DI_3"]
          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_switch": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_ls_switch": {
          "Channels": {
            "LSSW_1": ["DO_1"],
            "LSSW_2": ["DO_2"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_mpu6050": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_accel": {
          "Channels": {
            "dd_accel": ["MCAL_I2C"]
          },
          "Push": "TWI_ReadChannel",
          "Dependency": "mcal_i2c"
        },
        "dd_gyro": {
          "Channels": {
            "dd_gyro": ["MCAL_I2C"]
          },
          "Push": "TWI_ReadChannel",
          "Dependency": "mcal_i2c"
        }
      }
    },
    "dd_l298": {
      "git": "https://github.com/ML-ES-
Platform/dd_l298.git",
      "Path": "ESW/dd_l298/",
      "Layer": "ECAL",
      "Domain": "ACT",
      "Groups": {
        "dd_l298_pow": {
          "Channels": {
            "EN_A": ["PWM_1"],
            "EN_B": ["PWM_2"]
          },
          "Push": "PWM_WriteChannel",
          "Dependency": "mcal_pwm"
        },
        "dd_l298_dir": {
          "Channels": {
            "IN_1": ["DO_3"],
```

```
            "IN_2": ["DO_4"],
            "IN_3": ["DO_5"],
            "IN_4": ["DO_6"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dout"
        }
      }
    },
     "vd_com_serial": {
      "git": "https://github.com/ML-ES-
Platform/vd_com_serial.git",
      "Path": "ESW/vd_com_serial/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_serial": {
          "Channels": {
            "SERIAL": ["mcal_usart"]
          },
          "Dependency": "mcal_usart",
          "Pull": "USART_WriteByte"
        }
      }
    },
    "vd_com_tcp_ip": {
      "git": "https://github.com/ML-ES-
Platform/vd_com_tcp_ip.git",
      "Path": "ESW/vd_com_tcp_ip/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_tcp_ip": {
          "Channels": {
            "TCP_IP": ["mcal_spi"]
          },
          "Dependency": "mcal_spi",
          "Pull": "SPI_TransferByte"
        }
      }
    },

    "vd_act_dc_motor": {
      "git": "https://github.com/ML-ES-
Platform/vd_act_dc_motor.git",
      "Path": "ESW/vd_act_dc_motor/",
      "Layer": "SRV",
      "Domain": "ACT",
      "Groups": {
        "vd_dc_motor_pow": {
          "Channels": {
            "MOTOR_1":
["EN_A","IN_1","IN_2"],
            "MOTOR_2":
["EN_B","IN_3","IN_4"]
          },
          "Push": "DD_L298_SetPow",
          "Dependency": "dd_l298"
        }
      }
    },

    "vd_sns_encoder": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_encoder.git",
      "Path": "ESW/vd_sns_encoder/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_sns_encoder": {
          "Channels": {
            "PH_A": ["CNT_1"],
            "PH_B": ["CNT_2"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "vd_sns_anem": {
      "git": "https://github.com/ML-ES-
Platform/dd_sns_anemometer.git",
      "Path": "ESW/vd_sns_anemometer/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_anemometer": {
          "Channels": {
            "ANE_IN": ["CNT_5"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "vd_sns_acel": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_acelerometer.git",
      "Path": "ESW/vd_sns_acelerometer/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_acelerometer": {
          "Channels": {
            "ACCEL": ["dd_accel"]
          },
          "Dependency": "dd_mpu6050",
          "Pull": "I2C_ReadChannel"
        }
      }
    },
    "vd_sns_giro": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_giroscope.git",
      "Path": "ESW/vd_sns_giroscope/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_giroscope": {
          "Channels": {
            "GIRO": ["dd_gyro"]
          },
          "Dependency": "dd_mpu6050",
          "Pull": "I2C_ReadChannel"
        }
      }
    },
    "vd_sns_limit": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_limiter.git",
      "Path": "ESW/vd_sns_limiter/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_limiter": {
          "Channels": {
```

```
            "LIMITER": ["CNT_4"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },

    "vd_ui_led": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_led.git",
      "Path": "ESW/vd_ui_led/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_led": {
          "Channels": {
            "LED": ["LSSW_1"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },
     "vd_ui_button": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_button.git",
      "Path": "ESW/vd_ui_button/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_button": {
          "Channels": {
            "BUTTON": ["CNT_3"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "vd_ui_buzzer": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_buzzer.git",
      "Path": "ESW/vd_ui_buzzer/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_buzzer": {
          "Channels": {
            "BUZZ": ["LSSW_2"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },



     "os_time_trig": {
      "git": "https://github.com/ML-ES-
Platform/os_time_trig.git",
      "Path": "BSW/os_time_trig/",
      "Layer": "OS",
      "Domain": "OS"
    },
    "rte_com": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "tcp_ip": {
          "Channels": {
            "COM_TCP_IP": ["TCP_IP"]
          }
        },
        "bluetooth": {
          "Channels": {
            "COM_SERIAL": ["SERIAL"]
          }
        }
      }
    },
     "rte_sns": {
      "git": "https://github.com/ML-ES-
Platform/rte_sns.git",
      "Path": "BSW/rte_sns/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "limiter": {
          "Channels": {
            "SNS_LIMIT": ["LIMITER"]
          }
        },
        "anemometer": {
          "Channels": {
            "SNS_ANEM": ["ANE_IN"]
          }
        },
        "accelerometer": {
          "Channels": {
            "SNS_ACCEL": ["ACCEL"]
          }
        },
        "gyroscope": {
          "Channels": {
            "SNS_GIRO": ["GIRO"]
          }
        }
      }
    },
    "rte_act": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "motor_o": {
          "Channels": {
            "MOTOR_H": ["MOTOR_1"]
          }
        },
        "motor_v": {
          "Channels": {
            "MOTOR_V": ["MOTOR_2"]
          }
        }
      }
    },
    "rte_ui": {
      "git": "https://github.com/ML-ES-
Platform/rte_ui.git",
      "Path": "BSW/rte_ui/",
```

```
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "led": {
          "Channels": {
            "UI_LED": ["LED"]
          }
        },
        "button": {
          "Channels": {
            "UI_BUTTON": ["BUTTON"]
          }
        },
        "encoder": {
          "Channels": {
            "UI_ENCODER": ["PH_A","PH_B"]
          }
        },
        "buzzer": {
          "Channels": {
            "UI_BUZZER": ["BUZZ"]
          }
        }
      }
    },
    "comp_diagnostic": {
      "TypeName": "Diagnostic",
      "git": "https://github.com/ML-ES-
Platform/diagnostic.git",
      "Path": "ASW/diagnostic/",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "symptoms": {
          "Channels": {
            "wind_sympt": ["SNS_ANEM"] ,
            "limit_sympt":["SNS_LIMIT"]
          }
        },
        "diagnostic": {
          "Channels": {
            "diagnostic": ["wind_sympt",
"limit_sympt"]
          }
        }
      }
    },
    "comp_protect": {
      "TypeName": "Protect",
      "git": "https://github.com/ML-ES-
Platform/protect.git",
      "Path": "ASW/protect/",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "react": {
          "Channels": {
            "light_react": ["led_alert"] ,
            "sound_react": ["buzz_alert"],
            "vert_react":  ["vert_prot"],
            "horiz_react": ["horiz_prot"]
          }
        },
        "alert": {
          "Channels": {
            "led_alert": ["UI_LED"],
            "buzz_alert":["UI_BUZZER"]
          }
        },
        "protection": {
          "Channels": {
            "vert_prot": ["MOTOR_V"],
            "horiz_prot":["MOTOR_H"]
          }
        }
      }
    },
    "comp_miror_Pos": {
      "TypeName": "Miror_Pos",
      "git": "https://github.com/ML-ES-
Platform/mirror_pos.git",
      "Path": "ASW/mirror_pos/",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "angle_eval": {
          "Channels": {
            "angle_H":
["SNS_GIRO","SNS_ACCEL"],
            "angle_V":
["SNS_GIRO","SNS_ACCEL"]
          }
        }
      }
    },
    "comp_Ctrl": {
      "TypeName": "dev_ctrl",
      "git": "https://github.com/ML-ES-
Platform/dev_ctrl.git",
      "Path": "ASW/dev_ctrl/",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "ui_button": {
          "Channels": {
            "ctrl_ui": ["UI_BUTTON"]
          }
        },
        "ui_encoder": {
          "Channels": {
            "ctrl_ui": ["UI_ENCODER"]
          }
        },
        "ctrl_ui_buzz": {
          "Channels": {
            "diag_ui": ["sound_react"]
          }
        },
        "ctrl_ui_led": {
          "Channels": {
            "diag_ui": ["light_react"]
          }
        },
        "comm_serial": {
          "Channels": {
            "ctrl_com": ["COM_SERIAL"]
          }
        },
        "comm_tcp": {
          "Channels": {
            "ctrl_com": ["COM_TCP_IP"]
          }
        },
        "diag": {
          "Channels": {
```

```
            "ctrl_diag_lim":
["limit_diag"],
            "ctrl_diag_wind": ["wind_diag"]
          }
        },
        "prot_H": {
          "Channels": {
            "ctrl_prot_lim": ["horiz_react"]
          }
        },
        "prot_V": {
          "Channels": {
            "ctrl_prot_lim": ["vert_react"]
          }
        },
        "desired_angle_h": {
          "Channels": {
            "ctrl_ui":  ["ctrl_h"],
            "ctrl_com": ["ctrl_h"]
          }
        },
        "desired_angle_v": {
          "Channels": {
            "ctrl_com": ["ctrl_v"],
            "ctrl_ui":  ["ctrl_v"]
          }
        }
      }
    },
    "comp_ON_OFF_Ctrl": {
      "TypeName": "ON_OFF_Ctrl",
      "git": "https://github.com/ML-ES-
Platform/don_off_ctrl.git",
      "Path": "ASW/don_off_ctrl/",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "current_angle": {
          "Channels": {
            "ctrl_h": ["angle_H"],
            "ctrl_v": ["angle_V"]
          }
        },
        "ctrl_output": {
          "Channels": {
            "ctrl_h": ["MOTOR_H"],
            "ctrl_v": ["MOTOR_V"]
          }
        }
      }
    },
    "comp_heliostat": {
      "TypeName": "heliostat",
      "git": "https://github.com/ML-ES-
Platform/heliostat.git",
      "Path": "ASW/heliostat/",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "heliostat": {
          "Channels": {}
        },
        "actuator": {
          "Channels": {}
        },
        "sensor": {
          "Channels": {}
        },
        "communication": {
          "Channels": {}
        },
        "user_interaction": {
          "Channels": {}
        }
      }
    },
    "app_heliostat_demo": {
      "git": "https://github.com/ML-ES-
Platform/heliostat_demo.git",
      "Path": "ASW/heliostat/demo",
      "Layer": "APP_Heliostat",
      "Domain": "APP_Heliostat",
      "Groups": {
        "heliostat": {
          "Channels": {}
        }
      }
    }
  }
}
```

Cofiguratia JSON pentru sistem de control al camerei de uscare

```
{
  "Description": "Application Demo for Fruit
Dry control",
  "ProjectName": "fruit_dry",

  "git": "https://github.com/ML-ES-
Platform/fruit_dry.git",
  "Path": "ASW/fruit_dry/",

  "Components": {
    "MCU_IO": {
      "Layer": "MCU",
      "Groups": {
        "DI_PIN": {
          "Channels": {
            "D1": [],
            "D2": [],
            "D3": [],
            "D4": [],
            "D5": []
          }
        },
        "DO_PIN": {
          "Channels": {
            "D11": [],
            "D12": [],
            "D13": [],
            "D14": []
          }
        },
```

```
        "ADC_PIN": {
          "Channels": {
            "A1": [],
            "A2": []
          }
        },
        "DAC_PIN": {
          "Channels": {
            "A3": []
          }
        },
        "PWM_PIN": {
          "Channels": {
            "PWM1": []
          }
        },
        "ONE_WIRE_IF": {
          "Channels": {
          }
        },
        "I2C_IF": {
          "Channels": {
          }
        },
        "SPI_IF": {
          "Channels": {
          }
        },
        "USART_IF": {
          "Channels": {
            "usart": []
          }
        }
      }
    },
    "mcal_din_sns": {
      "git": "https://github.com/ML-ES-
Platform/mcal_din.git",
      "Path": "MCAL/mcal_din/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DI_4": ["D4"],
            "DI_5": ["D5"]
          }
        }
      }
    },
    "mcal_din_ui": {
      "git": "https://github.com/ML-ES-
Platform/mcal_din.git",
      "Path": "MCAL/mcal_din/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DI_1": ["D1"],
            "DI_2": ["D2"],
            "DI_3": ["D3"]
          }
        }
      }
    },
    "mcal_dout": {
      "git": "https://github.com/ML-ES-
Platform/mcal_dout.git",
      "Path": "MCAL/mcal_dout/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_din": {
          "Channels": {
            "DO_1": ["D11"],
            "DO_2": ["D12"],
            "DO_3": ["D13"],
            "DO_4": ["D14"]
          }
        }
      }
    },
    "mcal_adc": {
      "git": "https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_adc": {
          "Channels": {
            "ADC_1":["A1"],
            "ADC_2":["A2"]
          }
        }
      }
    },
    "mcal_dac": {
      "git": "https://github.com/ML-ES-
Platform/mcal_adc.git",
      "Path": "MCAL/mcal_adc/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_adc": {
          "Channels": {
            "DAC_1":["A3"]
          }
        }
      }
    },
    "mcal_pwm": {
      "git": "https://github.com/ML-ES-
Platform/mcal_pwm.git",
      "Path": "MCAL/mcal_pwm/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_pwm": {
          "Channels": {
            "PWM_1": ["PWM1"]
          }
        }
      }
    },
    "mcal_i2c": {
      "git": "https://github.com/ML-ES-
Platform/mcal_i2c.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_i2c": {
          "Channels": {
```

```
          }
        }
      }
    },
    "mcal_1w": {
      "git": "https://github.com/ML-ES-
Platform/mcal_1w.git",
      "Path": "MCAL/mcal_1w/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_1w": {
          "Channels": {
          }
        }
      }
    },
    "mcal_spi": {
      "git": "https://github.com/ML-ES-
Platform/mcal_spi.git",
      "Path": "MCAL/mcal_i2c/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_spi": {
          "Channels": {
          }
        }
      }
    },
    "mcal_usart": {
      "git": "https://github.com/ML-ES-
Platform/mcal_usart.git",
      "Path": "MCAL/mcal_usart/",
      "Layer": "MCAL",
      "Domain": "MCAL",
      "Groups": {
        "mcal_usart": {
          "Channels": {
            "mcal_usart": ["usart"]
          }
        }
      }
    },
    "dd_cnt_sns": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "SNS",
      "Groups": {
        "dd_contact": {
          "Channels": {
            "CNT_4": ["DI_4"],
            "CNT_5": ["DI_5"]
          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_cnt_ui": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_contact": {
          "Channels": {
            "CNT_1": ["DI_1"],
            "CNT_2": ["DI_2"],
            "CNT_3": ["DI_3"]

          },
          "Push": "DIO_ReadChannel",
          "Dependency": "mcal_dio"
        }
      }
    },
    "dd_switch": {
      "git": "https://github.com/ML-ES-
Platform/dd_contact.git",
      "Path": "ESW/dd_contact/",
      "Layer": "ECAL",
      "Domain": "UI",
      "Groups": {
        "dd_ls_switch": {
          "Channels": {
            "LSSW_1": ["DO_1"],
            "LSSW_2": ["DO_2"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dio"
        }
      }
    },

    "dd_ana_temp": {
      "git": "https://github.com/ML-ES-
Platform/dd_ana_temp.git",
      "Path": "ESW/dd_ana_temp/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "dd_ana_temp": {
          "Channels": {
            "dd_temp": ["ADC_1"]
          },
          "Dependency": "mcal_adc",
          "Pull": "ADC_ReadChannel"
        }
      }
    },
    "dd_ana_hum": {
      "git": "https://github.com/ML-ES-
Platform/dd_ana_hum.git",
      "Path": "ESW/dd_ana_hum/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "dd_ana_hum": {
          "Channels": {
            "dd_hum": ["ADC_2"]
          },
          "Dependency": "mcal_adc",
          "Pull": "ADC_ReadChannel"
        }
      }
    },

    "dd_l298": {
      "git": "https://github.com/ML-ES-
Platform/dd_l298.git",
      "Path": "ESW/dd_l298/",
      "Layer": "ECAL",
```

```
      "Domain": "ACT",
      "Groups": {
        "dd_l298_pow": {
          "Channels": {
            "EN_A": ["PWM_1"]
          },
          "Push": "PWM_WriteChannel",
          "Dependency": "mcal_pwm"
        },
        "dd_l298_dir": {
          "Channels": {
            "IN_1": ["DO_3"],
            "IN_2": ["DO_4"]
          },
          "Push": "DO_WriteChannel",
          "Dependency": "mcal_dout"
        }
      }
    },
    "dd_servo": {
      "git": "https://github.com/ML-ES-
Platform/dd_servo.git",
      "Path": "ESW/dd_servo/",
      "Layer": "ECAL",
      "Domain": "ACT",
      "Groups": {
        "DD_SERVO": {
          "Channels": {
            "DD_SRV_1": ["DAC_1"]
          },
          "Dependency": "dd_pca9685",
          "Push": "PCA9685_WriteChannel"
        }
      }
    },

    "vd_com_serial": {
      "git": "https://github.com/ML-ES-
Platform/vd_com_serial.git",
      "Path": "ESW/vd_com_serial/",
      "Layer": "SRV",
      "Domain": "COM",
      "Groups": {
        "vd_com_serial": {
          "Channels": {
            "SERIAL": ["mcal_usart"]
          },
          "Dependency": "mcal_usart",
          "Pull": "USART_WriteByte"
        }
      }
    },
    "vd_act_servo": {
      "git": "https://github.com/ML-ES-
Platform/vd_act_servo.git",
      "Path": "ESW/vd_act_servo/",
      "Layer": "SRV",
      "Domain": "ACT",
      "Groups": {
        "VD_ACT_SERVO": {
          "Channels": {
            "VD_SRV_1": ["DD_SRV_1"]
          },
          "Push": "DDSERVO_SetAngle"
        }
      }
    },
```

```
    "vd_act_dc_motor": {
      "git": "https://github.com/ML-ES-
Platform/vd_act_dc_motor.git",
      "Path": "ESW/vd_act_dc_motor/",
      "Layer": "SRV",
      "Domain": "ACT",
      "Groups": {
        "vd_dc_motor_pow": {
          "Channels": {
            "MTR_1": ["EN_A","IN_1","IN_2"]
          },
          "Push": "DD_L298_SetPow",
          "Dependency": "dd_l298"
        }
      }
    },

    "vd_sns_encoder": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_encoder.git",
      "Path": "ESW/vd_sns_encoder/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_sns_encoder": {
          "Channels": {
            "PH_A": ["CNT_1"],
            "PH_B": ["CNT_2"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "vd_sns_limiter": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_limiter.git",
      "Path": "ESW/vd_sns_limiter/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_limiter": {
          "Channels": {
            "LIM_L": ["CNT_4"],
            "LIM_R": ["CNT_5"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "vd_sns_temperature": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_temperature.git",
      "Path": "ESW/vd_sns_temperature/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_humidity": {
          "Channels": {
            "TEMP": ["dd_temp"]
          },
          "Dependency": "dd_ana_temp",
          "Pull":
"DD_ANAHUM_ReadTemperatuire"
        }
      }
    },
```

```
    "vd_sns_humidity": {
      "git": "https://github.com/ML-ES-
Platform/vd_sns_humidity.git",
      "Path": "ESW/vd_sns_humidity/",
      "Layer": "SRV",
      "Domain": "SNS",
      "Groups": {
        "vd_sns_humidity": {
          "Channels": {
            "HUM": ["dd_hum"]
          },
          "Dependency": "dd_ana_hum",
          "Pull": "DD_ANAHUM_ReadHumidity"
        }
      }
    },

    "vd_ui_led": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_led.git",
      "Path": "ESW/vd_ui_led/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_led": {
          "Channels": {
            "LED": ["LSSW_1"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },
     "vd_ui_button": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_button.git",
      "Path": "ESW/vd_ui_button/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_button": {
          "Channels": {
            "BUTTON": ["CNT_3"]
          },
          "Dependency": "mcal_din",
          "Pull": "DI_ReadChannel"
        }
      }
    },
    "vd_ui_buzzer": {
      "git": "https://github.com/ML-ES-
Platform/vd_ui_buzzer.git",
      "Path": "ESW/vd_ui_buzzer/",
      "Layer": "SRV",
      "Domain": "UI",
      "Groups": {
        "vd_ui_buzzer": {
          "Channels": {
            "BUZZ": ["LSSW_2"]
          },
          "Dependency": "mcal_dout",
          "Pull": "DO_WriteChannel"
        }
      }
    },


     "os_time_trig": {
      "git": "https://github.com/ML-ES-
Platform/os_time_trig.git",
      "Path": "BSW/os_time_trig/",
      "Layer": "OS",
      "Domain": "OS"
    },
    "rte_com": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "serial": {
          "Channels": {
            "COM_SER": ["SERIAL"]
          }
        }
      }
    },
     "rte_sns": {
      "git": "https://github.com/ML-ES-
Platform/rte_sns.git",
      "Path": "BSW/rte_sns/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "limiter": {
          "Channels": {
            "SNS_LIM_L": ["LIM_L"],
            "SNS_LIM_R": ["LIM_R"]
          }
        },
        "temperature": {
          "Channels": {
            "SNS_TEMP": ["TEMP"]
          }
        },
        "humidity": {
          "Channels": {
            "SNS_HUM": ["HUM"]
          }
        }
      }
    },
    "rte_act": {
      "git": "https://github.com/ML-ES-
Platform/rte_act.git",
      "Path": "BSW/rte_act/",
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "motor": {
          "Channels": {
            "DC_MTR": ["MTR_1"]
          }
        },
        "servo": {
          "Channels": {
            "ACT_SERVO": ["VD_SRV_1"]
          }
        }
      }
    },
    "rte_ui": {
      "git": "https://github.com/ML-ES-
Platform/rte_ui.git",
      "Path": "BSW/rte_ui/",
```

```
      "Layer": "RTE",
      "Domain": "RTE",
      "Groups": {
        "led": {
          "Channels": {
            "UI_LED": ["LED"]
          }
        },
        "button": {
          "Channels": {
            "UI_BTN": ["BUTTON"]
          }
        },
        "encoder": {
          "Channels": {
            "UI_ENC": ["PH_A","PH_B"]
          }
        },
        "buzzer": {
          "Channels": {
            "UI_BUZZ": ["BUZZ"]
          }
        }
      }
    },
    "comp_diagnostic": {
      "TypeName": "Diagnostic",
      "git": "https://github.com/ML-ES-
Platform/diagnostic.git",
      "Path": "ASW/diagnostic/",
      "Layer": "APP_Fruit_Dry",
      "Domain": "APP_Fruit_Dry",
      "Groups": {
        "symptoms": {
          "Channels": {
            "lim_l_sym": ["SNS_LIM_L"],
            "lim_r_sym": ["SNS_LIM_R"],
             "temp_sym": ["SNS_TEMP"],
              "hum_sym": ["SNS_HUM"]
          }
        },
        "diagnostic": {
          "Channels": {
            "diagnostic": ["lim_l_sym",
"lim_r_sym", "temp_sym",  "hum_sym"]
          }
        }
      }
    },

    "comp_HUM_Ctrl": {
      "TypeName": "ON_OFF_Ctrl",
      "git": "https://github.com/ML-ES-
Platform/on_off_ctrl.git",
      "Path": "ASW/on_off_ctrl/",
      "Layer": "APP_Fruit_Dry",
      "Domain": "APP_Fruit_Dry",
      "Groups": {
        "ctrl_hum": {
          "Channels": {
            "hum_ctrl": ["HUM_IN","HUM_OUT"]
          }
        },
        "ctrl_input": {
          "Channels": {
            "HUM_IN": ["SNS_HUM"]
          }
        },
        "ctrl_output": {
          "Channels": {
            "HUM_OUT": ["ACT_SERVO"]
          }
        }
      }
    },

    "comp_WGN_Ctrl": {
      "TypeName": "ON_OFF_Ctrl",
      "git": "https://github.com/ML-ES-
Platform/on_off_ctrl.git",
      "Path": "ASW/on_off_ctrl/",
      "Layer": "APP_Fruit_Dry",
      "Domain": "APP_Fruit_Dry",
      "Groups": {
        "ctrl_hum": {
          "Channels": {
            "wagon_feed":
["LIM_IN","WGN_OUT"]
          }
        },
        "ctrl_input": {
          "Channels": {
            "LIM_IN":
["SNS_LIM_L","SNS_LIM_R"]
          }
        },
        "ctrl_output": {
          "Channels": {
            "WGN_OUT": ["DC_MTR"]
          }
        }
      }
    },

    "comp_TEMP_Ctrl": {
      "TypeName": "ON_OFF_Ctrl",
      "git": "https://github.com/ML-ES-
Platform/on_off_ctrl.git",
      "Path": "ASW/on_off_ctrl/",
      "Layer": "APP_Fruit_Dry",
      "Domain": "APP_Fruit_Dry",
      "Groups": {
        "ctrl_temp": {
          "Channels": {
            "temp_ctrl":
["TEMP_IN","TEMP_OUT"]
          }
        },
        "ctrl_input": {
          "Channels": {
            "TEMP_IN": ["SNS_TEMP"]
          }
        },
        "ctrl_output": {
          "Channels": {
            "TEMP_OUT": ["ACT_SERVO"]
          }
        }
      }
    },
    "comp_fruit_dry_ctrl": {
      "TypeName": "heliostat",
      "git": "https://github.com/ML-ES-
Platform/heliostat.git",
      "Path": "ASW/heliostat/",
      "Layer": "APP_Fruit_Dry",
```

```
      "Domain": "APP_Fruit_Dry",
      "Groups": {
        "fruit_dry_ctrl": {
          "Channels": {
            "fruit_dry_ctrl": ["ui_ctrl",
"wagon_feed", "temp_ctrl",
"hum_ctrl","diagnostic"]
          }
        }
      }
    },
    "comp_ui_ctrl": {
      "git": "https://github.com/ML-ES-
Platform/heliostat_demo.git",
      "Path": "ASW/heliostat/demo",
```

```
      "Layer": "APP_Fruit_Dry",
      "Domain": "APP_Fruit_Dry",
      "Groups": {
        "ui_ctrl": {
          "Channels": {

"ui_ctrl":["UI_BTN" ,"UI_ENC" ,"UI_BUZZ" ,"U
I_LED","COM_SER"]
          }
        }
      }
    }
  }
}
```

Anexa 5. Configurații ale proiectelor generate în format C/C++

Declarații configuratii de interconectare compoinente

```
/**
 * @file arm_3dof_demo_cfg_gen.h
 * @author your name (you@domain.com)
 * @brief
 * @version 0.1
 * @date 2020-06-12
 *
 * @copyright Copyright (c) 2020
 *
 */
#ifndef _ARM_3DOF_DEMO_CONFIG_H_
#define _ARM_3DOF_DEMO_CONFIG_H_

#include "./PLF/platform_config.h"

#define DD_PCA9685_DEV_CONFIG

#define MCU_IO_CONFIG

#define MCAL_ADC_CONFIG
enum MCAL_ADC_Cnl_IdType {ADC_1, ADC_2,
ADC_3, MCAL_ADC_CHANNEL_NR_OF};
#include "MCAL/mcal_adc/mcal_adc.h"

#define MCAL_PWM_CONFIG
#include "MCAL/mcal_pwm/mcal_pwm.h"

#define MCAL_I2C_CONFIG
enum MCAL_I2C_Cnl_IdType {MCAL_I2C,
MCAL_I2C_CHANNEL_NR_OF};
#include "MCAL/mcal_i2c/mcal_i2c.h"

#define DD_POT_CONFIG
enum DD_POT_Cnl_IdType {POT_1, POT_2, POT_3,
DD_POT_CHANNEL_NR_OF};
#include "ESW/dd_pot/dd_pot.h"

#define DD_PCA9685_CONFIG
enum DD_PCA9685_Cnl_IdType
{DD_PCA9685_PWM_1, DD_PCA9685_PWM_2,
DD_PCA9685_PWM_3, DD_PCA9685_CHANNEL_NR_OF};
#define DD_PCA9685_PWM_FREQ 50
#include "ESW/dd_pca9685/dd_pca9685.h"

#define DD_SERVO_CONFIG
enum DD_SERVO_Cnl_IdType {DD_SERVO_1,
DD_SERVO_2, DD_SERVO_3,
DD_SERVO_CHANNEL_NR_OF};
#include "ESW/dd_servo/dd_servo.h"

#define VD_SNS_ANGLE_CONFIG
enum VD_SNS_ANGLE_Cnl_IdType {ANGSNS_1,
ANGSNS_2, ANGSNS_3,
VD_SNS_ANGLE_CHANNEL_NR_OF};
#include "ESW/vd_sns_angle/vd_sns_angle.h"

#define VD_ACT_SERVO_CONFIG
enum VD_ACT_SERVO_Cnl_IdType
{VD_ACT_SERVO_1, VD_ACT_SERVO_2,
VD_ACT_SERVO_3, VD_ACT_SERVO_CHANNEL_NR_OF};
#define SECOND_MS 1000.0
#define SERVO_TASK_REC 10
#define SERVO_TASK_OFFSET 100
#include "ESW/vd_act_servo/vd_act_servo.h"

#include "BSW/os_time_trig/os_time_trig.h"

#define RTE_ACT_CONFIG
enum SNS_ANG_FB_Cnl_IdType {ACT_SRV_1,
ACT_SRV_2, ACT_SRV_3,
SNS_ANG_FB_CHANNEL_NR_OF};
#include "BSW/rte_act/rte_act.h"

#define RTE_SNS_CONFIG
enum SNS_ANG_FB_Cnl_IdType {SNS_ANG_1,
SNS_ANG_2, SNS_ANG_3,
SNS_ANG_FB_CHANNEL_NR_OF};
#include "BSW/rte_sns/rte_sns.h"

#define ARM_3DOF_CONFIG
enum ARM_3DOF_Cnl_IdType {ARM_3DOF_1,
ARM_3DOF_CHANNEL_NR_OF};
enum ARM_3DOF_SERVO_Cnl_IdType {ACT_BASE,
ACT_L1, ACT_L2,
ARM_3DOF_SERVO_CHANNEL_NR_OF};
enum ARM_3DOF_SEGMENT_Cnl_IdType {SEG_BASE,
SEG_L1, SEG_L2,
ARM_3DOF_SEGMENT_CHANNEL_NR_OF};
enum ARM_3DOF_POSSET_Cnl_IdType
{POS_XYZ_SET, INV_CNM,
ARM_3DOF_POSSET_CHANNEL_NR_OF};
enum ARM_3DOF_ANGLESFB_Cnl_IdType
{ANG_FB_BASE, ANG_FB_L1, ANG_FB_L2,
ARM_3DOF_ANGLESFB_CHANNEL_NR_OF};
enum ARM_3DOF_POSFB_Cnl_IdType {POS_XYX_FB,
DIR_CNM, ARM_3DOF_POSFB_CHANNEL_NR_OF};
#include "ASW/arm_3dof/arm_3dof.h"

Std_ReturnType arm_3dof_demo_config(void);

#endif // _ARM_3DOF_DEMO_CONFIG_H_
```

Derfinitii funcții de interconecare componente proiect

```
/**
* @file arm_3dof_demo_cfg_gen.cpp
* @author your name (you@domain.com)
* @brief
```

```
* @version 0.1
* @date 2020-06-12
*
* @copyright Copyright (c) 2020
*
*/

#include "arm_3dof_demo_cfg_gen.h"

Std_ChannelIdType
DD_PCA9685_PIN_Group[DD_PCA9685_PIN_CHANNEL_
NR_OF] = {DD_PCA9685_PIN_1,
DD_PCA9685_PIN_2, DD_PCA9685_PIN_3};

Std_ChannelIdType
DD_PCA9685_DEV_Group[DD_PCA9685_DEV_CHANNEL_
NR_OF] = {DD_PCA9685_DEV};
Std_ChannelIdType
DD_PCA9685_DEV_DD_PCA9685_DEV_Cnl[3] =
{DD_PCA9685_PIN_1, DD_PCA9685_PIN_2,
DD_PCA9685_PIN_3};

Std_ChannelIdType
MCU_ADC_PIN_Group[MCU_ADC_PIN_CHANNEL_NR_OF]
= {A3, A4, A5};

Std_ChannelIdType
MCAL_ADC_Group[MCAL_ADC_CHANNEL_NR_OF] =
{ADC_1, ADC_2, ADC_3};
Std_ChannelIdType MCAL_ADC_ADC_1_Cnl[1] =
{A3};
Std_ChannelIdType MCAL_ADC_ADC_2_Cnl[1] =
{A4};
Std_ChannelIdType MCAL_ADC_ADC_3_Cnl[1] =
{A5};

Std_ChannelIdType
MCAL_I2C_Group[MCAL_I2C_CHANNEL_NR_OF] =
{MCAL_I2C};
Std_ChannelIdType MCAL_I2C_MCAL_I2C_Cnl[1] =
{DD_PCA9685_DEV};

Std_ChannelIdType
DD_POT_Group[DD_POT_CHANNEL_NR_OF] = {POT_1,
POT_2, POT_3};
Std_ChannelIdType DD_POT_POT_1_Cnl[1] =
{ADC_1};
Std_ChannelIdType DD_POT_POT_2_Cnl[1] =
{ADC_2};
Std_ChannelIdType DD_POT_POT_3_Cnl[1] =
{ADC_3};

Std_ChannelIdType
DD_PCA9685_Group[DD_PCA9685_CHANNEL_NR_OF] =
{DD_PCA9685_PWM_1, DD_PCA9685_PWM_2,
DD_PCA9685_PWM_3};
Std_ChannelIdType
DD_PCA9685_DD_PCA9685_PWM_1_Cnl[1] =
{MCAL_I2C};
Std_ChannelIdType
DD_PCA9685_DD_PCA9685_PWM_2_Cnl[1] =
{MCAL_I2C};
Std_ChannelIdType
DD_PCA9685_DD_PCA9685_PWM_3_Cnl[1] =
{MCAL_I2C};

Std_ChannelIdType
DD_SERVO_Group[DD_SERVO_CHANNEL_NR_OF] =
{DD_SERVO_1, DD_SERVO_2, DD_SERVO_3};
Std_ChannelIdType DD_SERVO_DD_SERVO_1_Cnl[1]
= {DD_PCA9685_PWM_1};
Std_ChannelIdType DD_SERVO_DD_SERVO_2_Cnl[1]
= {DD_PCA9685_PWM_2};
Std_ChannelIdType DD_SERVO_DD_SERVO_3_Cnl[1]
= {DD_PCA9685_PWM_3};

Std_ChannelIdType
VD_SNS_ANGLE_Group[VD_SNS_ANGLE_CHANNEL_NR_O
F] = {ANGSNS_1, ANGSNS_2, ANGSNS_3};
Std_ChannelIdType
VD_SNS_ANGLE_ANGSNS_1_Cnl[1] = {POT_1};
Std_ChannelIdType
VD_SNS_ANGLE_ANGSNS_2_Cnl[1] = {POT_2};
Std_ChannelIdType
VD_SNS_ANGLE_ANGSNS_3_Cnl[1] = {POT_3};

Std_ChannelIdType
VD_ACT_SERVO_Group[VD_ACT_SERVO_CHANNEL_NR_O
F] = {VD_ACT_SERVO_1, VD_ACT_SERVO_2,
VD_ACT_SERVO_3};
Std_ChannelIdType
VD_ACT_SERVO_VD_ACT_SERVO_1_Cnl[1] =
{DD_SERVO_1};
Std_ChannelIdType
VD_ACT_SERVO_VD_ACT_SERVO_2_Cnl[1] =
{DD_SERVO_2};
Std_ChannelIdType
VD_ACT_SERVO_VD_ACT_SERVO_3_Cnl[1] =
{DD_SERVO_3};

Std_ChannelIdType
SNS_ANG_FB_Group[SNS_ANG_FB_CHANNEL_NR_OF] =
{ACT_SRV_1, ACT_SRV_2, ACT_SRV_3};
Std_ChannelIdType
SNS_ANG_FB_ACT_SRV_1_Cnl[1] =
{VD_ACT_SERVO_1};
Std_ChannelIdType
SNS_ANG_FB_ACT_SRV_2_Cnl[1] =
{VD_ACT_SERVO_2};
Std_ChannelIdType
SNS_ANG_FB_ACT_SRV_3_Cnl[1] =
{VD_ACT_SERVO_3};

Std_ChannelIdType
SNS_ANG_FB_Group[SNS_ANG_FB_CHANNEL_NR_OF] =
{SNS_ANG_1, SNS_ANG_2, SNS_ANG_3};
Std_ChannelIdType
SNS_ANG_FB_SNS_ANG_1_Cnl[1] = {ANGSNS_1};
Std_ChannelIdType
SNS_ANG_FB_SNS_ANG_2_Cnl[1] = {ANGSNS_2};
Std_ChannelIdType
SNS_ANG_FB_SNS_ANG_3_Cnl[1] = {ANGSNS_3};

Std_ChannelIdType
ARM_3DOF_Group[ARM_3DOF_CHANNEL_NR_OF] =
{ARM_3DOF_1};
Std_ChannelIdType ARM_3DOF_ARM_3DOF_1_Cnl[2]
= {POS_XYZ_SET, POS_XYX_FB};

Std_ChannelIdType
ARM_3DOF_SERVO_Group[ARM_3DOF_SERVO_CHANNEL_
NR_OF] = {ACT_BASE, ACT_L1, ACT_L2};
```

```
Std_ChannelIdType
ARM_3DOF_SERVO_ACT_BASE_Cnl[1] =
{ACT_SRV_1};
Std_ChannelIdType
ARM_3DOF_SERVO_ACT_L1_Cnl[1] = {ACT_SRV_2};
Std_ChannelIdType
ARM_3DOF_SERVO_ACT_L2_Cnl[1] = {ACT_SRV_3};

Std_ChannelIdType
ARM_3DOF_SEGMENT_Group[ARM_3DOF_SEGMENT_CHAN
NEL_NR_OF] = {SEG_BASE, SEG_L1, SEG_L2};
Std_ChannelIdType
ARM_3DOF_SEGMENT_SEG_BASE_Cnl[2] = {INV_CNM,
DIR_CNM};
Std_ChannelIdType
ARM_3DOF_SEGMENT_SEG_L1_Cnl[2] = {INV_CNM,
DIR_CNM};
Std_ChannelIdType
ARM_3DOF_SEGMENT_SEG_L2_Cnl[2] = {INV_CNM,
DIR_CNM};

Std_ChannelIdType
ARM_3DOF_POSSET_Group[ARM_3DOF_POSSET_CHANNE
L_NR_OF] = {POS_XYZ_SET, INV_CNM};
Std_ChannelIdType
ARM_3DOF_POSSET_POS_XYZ_SET_Cnl[1] =
{INV_CNM};
Std_ChannelIdType
ARM_3DOF_POSSET_INV_CNM_Cnl[3] = {ACT_BASE,
ACT_L1, ACT_L2};

Std_ChannelIdType
ARM_3DOF_ANGLESFB_Group[ARM_3DOF_ANGLESFB_CH
ANNEL_NR_OF] = {ANG_FB_BASE, ANG_FB_L1,
ANG_FB_L2};
Std_ChannelIdType
ARM_3DOF_ANGLESFB_ANG_FB_BASE_Cnl[1] =
{SNS_ANG_1};
Std_ChannelIdType
ARM_3DOF_ANGLESFB_ANG_FB_L1_Cnl[1] =
{SNS_ANG_2};
Std_ChannelIdType
ARM_3DOF_ANGLESFB_ANG_FB_L2_Cnl[1] =
{SNS_ANG_3};

Std_ChannelIdType
ARM_3DOF_POSFB_Group[ARM_3DOF_POSFB_CHANNEL_
NR_OF] = {POS_XYX_FB, DIR_CNM};
Std_ChannelIdType
ARM_3DOF_POSFB_POS_XYX_FB_Cnl[1] =
{DIR_CNM};
Std_ChannelIdType
ARM_3DOF_POSFB_DIR_CNM_Cnl[3] =
{ANG_FB_BASE, ANG_FB_L1, ANG_FB_L2};

Std_ReturnType arm_3dof_demo_config(void)
{
  Serial.begin(115200);
  Serial.println("Application Demo for 3-DOF
Robotic arm  control");
  Std_ReturnType error = E_OK;

  error += MCAL_ADC_ChannelSetup(ADC_1, A3);
  error += MCAL_ADC_ChannelSetup(ADC_2, A4);
  error += MCAL_ADC_ChannelSetup(ADC_3, A5);
  Serial.print(" Group: mcal_adc configured
- Error : ");
  Serial.println(error);
```

```
  error += MCAL_I2C_ChannelSetup(MCAL_I2C,
DD_PCA9685_DEV);
  Serial.print(" Group: mcal_i2c configured
- Error : ");
  Serial.println(error);

  error += DD_POT_ChannelSetup(POT_1,
ADC_1);
  error += DD_POT_ChannelSetup(POT_2,
ADC_2);
  error += DD_POT_ChannelSetup(POT_3,
ADC_3);

  error += DD_POT_SetPullMethod(POT_1,
ADC_ReadChannel);
  error += DD_POT_SetPullMethod(POT_2,
ADC_ReadChannel);
  error += DD_POT_SetPullMethod(POT_3,
ADC_ReadChannel);
  Serial.print(" Group: dd_pot configured -
Error : ");
  Serial.println(error);

  error +=
DD_PCA9685_ChannelSetup(DD_PCA9685_PWM_1,
MCAL_I2C);
  error +=
DD_PCA9685_ChannelSetup(DD_PCA9685_PWM_2,
MCAL_I2C);
  error +=
DD_PCA9685_ChannelSetup(DD_PCA9685_PWM_3,
MCAL_I2C);
  Serial.print(" Group: dd_pca9685
configured - Error : ");
  Serial.println(error);


  error += DD_SERVO_ChannelSetup(DD_SERVO_1,
DD_PCA9685_PWM_1);
  error += DD_SERVO_ChannelSetup(DD_SERVO_2,
DD_PCA9685_PWM_2);
  error += DD_SERVO_ChannelSetup(DD_SERVO_3,
DD_PCA9685_PWM_3);

    error +=
DD_SERVO_SetPushMethod(DD_SERVO_1,
DD_PCA9685_SetPulseValue);
    error +=
DD_SERVO_SetPushMethod(DD_SERVO_2,
DD_PCA9685_SetPulseValue);
    error +=
DD_SERVO_SetPushMethod(DD_SERVO_3,
DD_PCA9685_SetPulseValue);
  Serial.print(" Group: dd_servo configured
- Error : ");
  Serial.println(error);


  error +=
VD_SNS_ANGLE_ChannelSetup(ANGSNS_1, POT_1);
  error +=
VD_SNS_ANGLE_ChannelSetup(ANGSNS_2, POT_2);
  error +=
VD_SNS_ANGLE_ChannelSetup(ANGSNS_3, POT_3);
```

```
  error +=
VD_SNS_ANGLE_SetPullMethod(ANGSNS_1,
DD_POT_GetPositionValue);
  error +=
VD_SNS_ANGLE_SetPullMethod(ANGSNS_2,
DD_POT_GetPositionValue);
  error +=
VD_SNS_ANGLE_SetPullMethod(ANGSNS_3,
DD_POT_GetPositionValue);
  Serial.print(" Group: vd_sns_angle
configured - Error : ");
  Serial.println(error);


  error +=
VD_ACT_SERVO_ChannelSetup(VD_ACT_SERVO_1,
DD_SERVO_1);
  error +=
VD_ACT_SERVO_ChannelSetup(VD_ACT_SERVO_2,
DD_SERVO_2);
  error +=
VD_ACT_SERVO_ChannelSetup(VD_ACT_SERVO_3,
DD_SERVO_3);

    error +=
VD_ACT_SERVO_SetPushMethod(VD_ACT_SERVO_1,
DD_SERVO_SetAngle);
    error +=
VD_ACT_SERVO_SetPushMethod(VD_ACT_SERVO_2,
DD_SERVO_SetAngle);
    error +=
VD_ACT_SERVO_SetPushMethod(VD_ACT_SERVO_3,
DD_SERVO_SetAngle);
  Serial.print(" Group: vd_act_servo
configured - Error : ");
  Serial.println(error);

  error += RTE_ACT_ChannelSetup(ACT_SRV_1,
VD_ACT_SERVO_1);
  error += RTE_ACT_ChannelSetup(ACT_SRV_2,
VD_ACT_SERVO_2);
  error += RTE_ACT_ChannelSetup(ACT_SRV_3,
VD_ACT_SERVO_3);
  Serial.print(" Group: sns_ang_fb
configured - Error : ");
  Serial.println(error);


  error += RTE_SNS_ChannelSetup(SNS_ANG_1,
ANGSNS_1);
  error += RTE_SNS_ChannelSetup(SNS_ANG_2,
ANGSNS_2);
  error += RTE_SNS_ChannelSetup(SNS_ANG_3,
ANGSNS_3);
  Serial.print(" Group: sns_ang_fb
configured - Error : ");
  Serial.println(error);


  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
POS_XYZ_SET);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
POS_XYX_FB);
  Serial.print(" Group: ARM_3DOF configured
- Error : ");
  Serial.println(error);
```

```
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
ACT_BASE, ACT_SRV_1);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
ACT_L1, ACT_SRV_2);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
ACT_L2, ACT_SRV_3);

    error +=
ARM_3DOF_SetPushMethod(ACT_BASE,
VD_ACT_SERVO_SetAngleValue);
    error += ARM_3DOF_SetPushMethod(ACT_L1,
VD_ACT_SERVO_SetAngleValue);
    error += ARM_3DOF_SetPushMethod(ACT_L2,
VD_ACT_SERVO_SetAngleValue);
  Serial.print(" Group: ARM_3DOF_Servo
configured - Error : ");
  Serial.println(error);

  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
SEG_BASE, INV_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
SEG_BASE, DIR_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
SEG_L1, INV_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
SEG_L1, DIR_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
SEG_L2, INV_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
SEG_L2, DIR_CNM);
  Serial.print(" Group: arm_3dof_Segment
configured - Error : ");
  Serial.println(error);

  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
POS_XYZ_SET, INV_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
INV_CNM, ACT_BASE);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
INV_CNM, ACT_L1);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
INV_CNM, ACT_L2);

  error +=
ARM_3DOF_SetPullMethod(POS_XYZ_SET,
REV_CNM_GetPos);
  error += ARM_3DOF_SetPullMethod(INV_CNM,
REV_CNM_GetPos);
  Serial.print(" Group: ARM_3DOF_PosSet
configured - Error : ");
  Serial.println(error);

  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
ANG_FB_BASE, SNS_ANG_1);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
ANG_FB_L1, SNS_ANG_2);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
ANG_FB_L2, SNS_ANG_3);

  error +=
ARM_3DOF_SetPullMethod(ANG_FB_BASE,
ANGSNS_GetAngle);
  error += ARM_3DOF_SetPullMethod(ANG_FB_L1,
ANGSNS_GetAngle);
  error += ARM_3DOF_SetPullMethod(ANG_FB_L2,
ANGSNS_GetAngle);
  Serial.print(" Group: ARM_3DOF_AnglesFB
configured - Error : ");
```

```
  Serial.println(error);

  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
POS_XYX_FB, DIR_CNM);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
DIR_CNM, ANG_FB_BASE);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
DIR_CNM, ANG_FB_L1);
  error += ARM_3DOF_ChannelSetup(ARM_3DOF_1,
DIR_CNM, ANG_FB_L2);
```

```
  error +=
ARM_3DOF_SetPullMethod(POS_XYX_FB,
REV_CNM_GetPos);
  error += ARM_3DOF_SetPullMethod(DIR_CNM,
REV_CNM_GetPos);
  Serial.print(" Group: ARM_3DOF_PosFB
configured - Error : ");
  Serial.println(error);

  return error;
}
```

Anexa 6. Componente de platformă generate

Comparație versiune componenta generata, cu versiunea completata și adaptata proiectului.

Componenta adaptată la Proiect – fișier Header.	*Componenta generată de platforma – fișier Header*

/*
 * vd_act_servo.h
 *
 * Created on: May 13, 2020
 *   Author: Andrei Bragarenco
 */
 	
#ifndef _VD_ACT_SERVO_H_
#define _VD_ACT_SERVO_H_
 	
#include "vd_act_servo_cfg.h"
 	
#ifndef VD_ACT_SERVO_CONFIG
enum VD_ACT_SERVO_Cnl_IdType
{VD_ACT_SERVO_CHANNEL_NR_OF = 0 };
#endif
 	
#define SERVO_IDL 1
#define SERVO_ACC 2
#define SERVO_BRK 3
#define SERVO_NAV 4
#define SERVO_FWD (1)
#define SERVO_STP (0)
#define SERVO_BWD (-1)
 	
#define SECOND_MS 1000.0
 	
typedef struct VD_ACT_SERVO_ChannelType_t {
 	
Std_ChannelIdType linkChannelId = 0;
 	
int8_t servoNavState = SERVO_IDL;
int8_t servoDirection = SERVO_STP;
 	
// GRD
Std_PhyDataType curentAngle = 0;
Std_PhyDataType startAngle = 0;
Std_PhyDataType targetAngle = 0;
Std_PhyDataType deltaAngle = 0;
 	
Std_PhyDataType ANGLE_MIN = 0;
Std_PhyDataType ANGLE_MAX = 180;
 	
// GRD/SEC
Std_PhyDataType currentSpeed = 0;
Std_PhyDataType accCurrentSpeed = 0;
Std_PhyDataType decCurrentSpeed = 0;
 	
Std_PhyDataType acceleratePhy = (60.0);
Std_PhyDataType accelerateRec = (0);
 	
Std_PhyDataType SPEED_MIN = (5.0);
Std_PhyDataType SPEED_MAX = (90.0);
 	
Std_PhySetterType SetCnlAngleValue = NULL;
Std_PhyGetterType GetCnlAngleValue = NULL;
 	
int8_t servo_run_recurency = 10;
 	
} VD_ACT_SERVO_ChannelType;
	/**
	 * @file vd_act_servo.h
	 * @author Admin (you@domain.com)
	* @date 29-Jan-2023
	 *
	 */
	
	#ifndef _VD_ACT_SERVO_H_
	#define _VD_ACT_SERVO_H_
	
	#include "vd_act_servo_cfg.h"
	
	#ifndef VD_ACT_SERVO_CONFIG
	enum VD_ACT_SERVO_Cnl_IdType
	{ VD_ACT_SERVO_CHANNEL_NR_OF = 0 };
	#endif
	
	typedef struct VD_ACT_SERVO_ChannelType_t {
	
	Std_ChannelIdType linkChannelId = 0;
	
	Std_PhyDataType input_value = 0;
	Std_PhyDataType IN_MIN = 0;
	Std_PhyDataType IN_MAX = 100;
	
	Std_PhyDataType output_value = 0;
	Std_PhyDataType OUT_MIN = 0;
	Std_PhyDataType OUT_MAX = 100;
	
	Std_PhySetterType SetLinkChannelValue = NULL;
	Std_PhyGetterType GetLinkChannelValue = NULL;
	
	int8_t run_recurency = 10;

```
VD_ACT_SERVO_ChannelType*
VD_ACT_SERVO_GetChannelRef(Std_ChannelIdType
channelId);
Std_ReturnType
VD_ACT_SERVO_SetupChannel(Std_ChannelIdType
channelId, Std_ChannelIdType linkChannelId);
Std_ReturnType
VD_ACT_SERVO_SetupGroupChannels(Std_ChannelId
Type *srcIds, Std_ChannelIdType *targhetIds,
uint8_t nr_of_channels);

Std_ReturnType
VD_ACT_SERVO_SetPushMethod(Std_ChannelIdType
channelId,  Std_PhySetterType pushMethod);
Std_ReturnType
VD_ACT_SERVO_SetGroupPushMethod(Std_ChannelId
Type *srcIds, Std_PhySetterType pushMethod,
uint8_t nr_of_channels);









Std_ReturnType
VD_ACT_SERVO_SetAngleValue(Std_ChannelIdType
channelId, Std_PhyDataType value);
Std_ReturnType
VD_ACT_SERVO_SetAngleValueByRef(VD_ACT_SERVO_
ChannelType *channelRef, Std_PhyDataType
angle);

Std_ReturnType
VD_ACT_SERVO_AngleAdd(Std_ChannelIdType
channelId, Std_PhyDataType angle);
Std_ReturnType
VD_ACT_SERVO_AngleAddByRef(VD_ACT_SERVO_Chann
elType *channelRef, Std_PhyDataType angle);

Std_PhyDataType
VD_ACT_SERVO_GetAngleValue(Std_ChannelIdType
channelId);
Std_PhyDataType
VD_ACT_SERVO_GetAngleValueByRef(VD_ACT_SERVO_
ChannelType *channelRef);

Std_PhyDataType
VD_ACT_SERVO_GetAngleOutValue(Std_ChannelIdTy
pe channelId);
Std_PhyDataType
VD_ACT_SERVO_GetAngleOutValueByRef(VD_ACT_SER
VO_ChannelType *channelRef);

Std_ReturnType
VD_ACT_SERVO_SetAngleLimits(VD_ACT_SERVO_Chan
nelType *channelRef, Std_PhyDataType IN_MIN,
Std_PhyDataType IN_MAX);
Std_ReturnType
VD_ACT_SERVO_SetSpeedLimits(VD_ACT_SERVO_Chan
nelType *channelRef, Std_PhyDataType OUT_MIN,
Std_PhyDataType OUT_MAX);
```

```
} VD_ACT_SERVO_ChannelType;


VD_ACT_SERVO_ChannelType*
VD_ACT_SERVO_GetChannelRef(Std_ChannelIdTyp
e channelId);
Std_ReturnType
VD_ACT_SERVO_SetupChannel(Std_ChannelIdType
channelId, Std_ChannelIdType
linkChannelId);
Std_ReturnType
VD_ACT_SERVO_SetupGroupChannels(Std_Channel
IdType *srcIds, Std_ChannelIdType
*targhetIds, uint8_t nr_of_channels);

Std_ReturnType
VD_ACT_SERVO_SetPushMethod(Std_ChannelIdTyp
e channelId,  Std_PhySetterType
pushMethod);
Std_ReturnType
VD_ACT_SERVO_SetGroupPushMethod(Std_Channel
IdType *srcIds, Std_PhySetterType
pushMethod, uint8_t nr_of_channels);

Std_ReturnType
VD_ACT_SERVO_SetPullMethod(Std_ChannelIdTyp
e channelId,  Std_PhyGetterType
pullMethod);
Std_ReturnType
VD_ACT_SERVO_SetGroupPullMethod(Std_Channel
IdType *srcIds, Std_PhyGetterType
pullMethod, uint8_t nr_of_channels);

Std_ReturnType
VD_ACT_SERVO_SetInputValue(Std_ChannelIdTyp
e channelId, Std_PhyDataType value);
Std_ReturnType
VD_ACT_SERVO_SetInputValueByRef(VD_ACT_SERV
O_ChannelType *channelRef, Std_PhyDataType
value);

Std_ReturnType
VD_ACT_SERVO_SetOutputValue(Std_ChannelIdTy
pe channelId, Std_PhyDataType value);
Std_ReturnType
VD_ACT_SERVO_SetOutputValueByRef(VD_ACT_SER
VO_ChannelType *channelRef, Std_PhyDataType
value);

Std_PhyDataType
VD_ACT_SERVO_GetInputValue(Std_ChannelIdTyp
e channelId);
Std_PhyDataType
VD_ACT_SERVO_GetInputValueByRef(VD_ACT_SERV
O_ChannelType *channelRef);

Std_PhyDataType
VD_ACT_SERVO_GetOutputValue(Std_ChannelIdTy
pe channelId);
Std_PhyDataType
VD_ACT_SERVO_GetOutputValueByRef(VD_ACT_SER
VO_ChannelType *channelRef);

Std_ReturnType
VD_ACT_SERVO_SetInputLimits(VD_ACT_SERVO_Ch
annelType *channelRef, Std_PhyDataType
IN_MIN, Std_PhyDataType IN_MAX);
```

```
Std_ReturnType
VD_ACT_SERVO_ChannelRunnable(void*
parameters);
Std_ReturnType
VD_ACT_SERVO_SetChannelRecurency(Std_ChannelI
dType channelId, int recurency);
Std_ReturnType
VD_ACT_SERVO_SetGroupRecurency(Std_ChannelIdT
ype *srcIds, int recurency, uint8_t
nr_of_channels);

Std_ReturnType
VD_ACT_SERVO_AngleProcess(Std_ChannelIdType
channelId);
Std_ReturnType
VD_ACT_SERVO_AngleProcess(VD_ACT_SERVO_Channe
lType *channelRef);
Std_ReturnType
VD_ACT_SERVO_SpeedReset(Std_ChannelIdType
channelId);


#endif /* _VD_ACT_SERVO_H_ */
```

```
Std_ReturnType
VD_ACT_SERVO_SetOutputLimits(VD_ACT_SERVO_C
hannelType *channelRef, Std_PhyDataType
OUT_MIN, Std_PhyDataType OUT_MAX);

Std_ReturnType
VD_ACT_SERVO_SetupComponent(void*
parameters);
Std_ReturnType
VD_ACT_SERVO_ChannelRunnable(void*
parameters);
Std_ReturnType
VD_ACT_SERVO_SetChannelRecurency(Std_Channe
lIdType channelId, int recurency);
Std_ReturnType
VD_ACT_SERVO_SetGroupRecurency(Std_ChannelI
dType *srcIds, int recurency, uint8_t
nr_of_channels);


#endif // _VD_ACT_SERVO_H_
```

Comparație versiune componenta generata, cu versiunea completata și adaptata proiectului.

Componenta adaptată la Proiect – fișier Sursă.	***Componenta generată de platforma – fișier Sursă***

`/*`
` * vd_act_servo.cpp`
` *`
` *  Created on: May 13, 2020`
` *    Author: Andrei Bragarenco`
` */`
 	
`#include "vd_act_servo.h"`
 	
`#ifdef PLATFORM_CONFIG_ENABLE`
`extern VD_ACT_SERVO_ChannelType`
`VD_ACT_SERVO_Channels[VD_ACT_SERVO_CHANNEL_NR_OF];`
 	
`#else`
`VD_ACT_SERVO_ChannelType`
`VD_ACT_SERVO_Channels[VD_ACT_SERVO_CHANNEL_NR_OF];`
 	
`#endif`
 	
`VD_ACT_SERVO_ChannelType*`
`VD_ACT_SERVO_GetChannelRef(Std_ChannelIdType`
`channelId)`
`{`
`  VD_ACT_SERVO_ChannelType *channelRef;`
`  if (channelId < VD_ACT_SERVO_CHANNEL_NR_OF) {`
`    channelRef =`
`&VD_ACT_SERVO_Channels[channelId];`
`  } else {`
`    channelRef = NULL;`
`  }`
`  return channelRef;`
`}`
 	
`Std_ReturnType`
`VD_ACT_SERVO_SetupChannel(Std_ChannelIdType`
`channelId, Std_ChannelIdType linkChannelId)`
`{`
`  Std_ReturnType error = E_NOT_OK;`
	`/**`
	` * @file vd_act_servo.cpp`
	` * @author Admin (you@domain.com)`
	` * @date 29-Jan-2023`
	` *`
	` */`
	
	`#include "vd_act_servo.h"`
	
	`#ifdef PLATFORM_CONFIG_ENABLE`
	`extern VD_ACT_SERVO_ChannelType`
	`VD_ACT_SERVO_Channels[VD_ACT_SERVO_CHANNEL_NR_OF]`
	`;`
	`#else`
	`VD_ACT_SERVO_ChannelType`
	`VD_ACT_SERVO_Channels[VD_ACT_SERVO_CHANNEL_NR_OF]`
	`;`
	`#endif`
	
	`VD_ACT_SERVO_ChannelType*`
	`VD_ACT_SERVO_GetChannelRef(Std_ChannelIdType`
	`channelId)`
	`{`
	`  VD_ACT_SERVO_ChannelType *channelRef;`
	`  if (channelId < VD_ACT_SERVO_CHANNEL_NR_OF) {`
	`    channelRef =`
	`&VD_ACT_SERVO_Channels[channelId];`
	`  } else {`
	`    channelRef = NULL;`
	`  }`
	`  return channelRef;`
	`}`
	
	`Std_ReturnType`
	`VD_ACT_SERVO_SetupChannel(Std_ChannelIdType`
	`channelId, Std_ChannelIdType linkChannelId)`
	`{`
	`  Std_ReturnType error = E_NOT_OK;`

```
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef != NULL) {
    channelRef->linkChannelId = linkChannelId;
    error = E_OK;
  } else {
    error = E_NOT_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetupGroupChannels(Std_ChannelIdType
*srcIds, Std_ChannelIdType *targhetIds, uint8_t
nr_of_channels)
{
  Std_ReturnType error = E_OK;
  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    Std_ChannelIdType targhetId = targhetIds[i];
    error += VD_ACT_SERVO_SetupChannel(srcId,
targhetId);
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetPushMethod(Std_ChannelIdType
channelId,  Std_PhySetterType pushMethod)
{
  Std_ReturnType error = E_NOT_OK;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef!=NULL) {
    channelRef->SetCnlAngleValue = pushMethod;
    error = E_OK;
  } else {
    error = E_NOT_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetGroupPushMethod(Std_ChannelIdType
*srcIds, Std_PhySetterType pushMethod, uint8_t
nr_of_channels)
{
  Std_ReturnType error = E_OK;

  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    error += VD_ACT_SERVO_SetPushMethod(srcId,
pushMethod);
  }
  return error;
}
```

```
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef != NULL) {
    channelRef->linkChannelId = linkChannelId;
    error = E_OK;
  } else {
    error = E_NOT_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetupGroupChannels(Std_ChannelIdType
*srcIds, Std_ChannelIdType *targhetIds, uint8_t
nr_of_channels)
{
  Std_ReturnType error = E_OK;
  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    Std_ChannelIdType targhetId = targhetIds[i];
    error += VD_ACT_SERVO_SetupChannel(srcId,
targhetId);
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetPushMethod(Std_ChannelIdType
channelId,  Std_PhySetterType pushMethod)
{
  Std_ReturnType error = E_NOT_OK;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef!=NULL) {
    channelRef->SetLinkChannelValue = pushMethod;
    error = E_OK;
  } else {
    error = E_NOT_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetGroupPushMethod(Std_ChannelIdType
*srcIds, Std_PhySetterType pushMethod, uint8_t
nr_of_channels)
{
  Std_ReturnType error = E_OK;

  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    error += VD_ACT_SERVO_SetPushMethod(srcId,
pushMethod);
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetPullMethod(Std_ChannelIdType
channelId,  Std_PhyGetterType pullMethod)
{
  Std_ReturnType error = E_NOT_OK;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef != NULL) {
    channelRef->GetLinkChannelValue = pullMethod;
    error = E_OK;
  } else {
```

```
Std_ReturnType
VD_ACT_SERVO_SetAngleValue(Std_ChannelIdType
channelId, Std_PhyDataType angle) {

  Std_ReturnType error = E_NOT_OK;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef != NULL) {
    error =
VD_ACT_SERVO_SetAngleValueByRef(channelRef,
angle);
  }
  return error;
}

Std_ReturnType
VD_ACT_SERVO_SetAngleValueByRef(VD_ACT_SERVO_Chann
elType *channelRef, Std_PhyDataType angle)
{
  Std_ReturnType error = E_NOT_OK;
  if (channelRef != NULL) {

    if (angle > channelRef->ANGLE_MAX)
      angle = channelRef->ANGLE_MAX; //saturatie
    if (angle < channelRef->ANGLE_MIN)
      angle = channelRef->ANGLE_MIN; //saturatie

    channelRef->targetAngle = angle;
    channelRef->accCurrentSpeed =
channelRef->currentSpeed;
    error = E_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_AngleAdd(Std_ChannelIdType channelId,
Std_PhyDataType angle)
{
  Std_ReturnType error = E_NOT_OK;
     VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if (channelRef != NULL) {
    error =
VD_ACT_SERVO_AngleAddByRef(channelRef, angle);

  }
  return error;
}
```

```
    error = E_NOT_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetGroupPullMethod(Std_ChannelIdType
*srcIds, Std_PhyGetterType pullMethod, uint8_t
nr_of_channels)
{
  Std_ReturnType error = E_OK;

  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    error += VD_ACT_SERVO_SetPullMethod(srcId,
pullMethod);
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetInputValue(Std_ChannelIdType
channelId, Std_PhyDataType value)
{
  Std_ReturnType error = E_NOT_OK;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  if (channelRef != NULL) {
    error =
VD_ACT_SERVO_SetInputValueByRef(channelRef,
value);
  }
  return error;
}

Std_ReturnType
VD_ACT_SERVO_SetInputValueByRef(VD_ACT_SERVO_Chan
nelType *channelRef, Std_PhyDataType value)
{
  Std_ReturnType error = E_NOT_OK;
  if (channelRef != NULL) {

    if (value > channelRef->IN_MAX)
      value = channelRef->IN_MAX; //saturatie
    if (value < channelRef->IN_MIN)
      value = channelRef->IN_MIN; //saturatie

    channelRef->input_value = value;


    error = E_OK;
  }
  return error;
}


Std_ReturnType
VD_ACT_SERVO_SetOutputValue(Std_ChannelIdType
channelId, Std_PhyDataType value)
{
  Std_ReturnType error = E_NOT_OK;
     VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if (channelRef != NULL) {
    error =
VD_ACT_SERVO_SetOutputValueByRef(channelRef,
value);
  }
  return error;
}
```

```
Std_ReturnType
VD_ACT_SERVO_AngleAddByRef(VD_ACT_SERVO_ChannelTyp
e *channelRef, Std_PhyDataType angle)
{
  Std_ReturnType error = E_NOT_OK;
  Std_PhyDataType target_angle = 0;

  if (channelRef != NULL){
    target_angle = channelRef->targetAngle +
angle;
    error =
VD_ACT_SERVO_SetAngleValueByRef(channelRef,
target_angle);
  }

  return error;
}


Std_PhyDataType
VD_ACT_SERVO_GetAngleValue(Std_ChannelIdType
channelId)
{
  Std_PhyDataType angle_value = 0;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if (channelRef != NULL) {
    angle_value =
VD_ACT_SERVO_GetAngleValueByRef(channelRef);
  }
  return angle_value;
}


Std_PhyDataType
VD_ACT_SERVO_GetAngleValueByRef(VD_ACT_SERVO_Chann
elType *channelRef)
{
  Std_PhyDataType angle_value = 0;
  if (channelRef != NULL) {
    angle_value =  channelRef->targetAngle;
  }
  return angle_value;
}


Std_PhyDataType
VD_ACT_SERVO_GetAngleOutValue(Std_ChannelIdType
channelId)
{
  Std_PhyDataType output_value = 0;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if (channelRef != NULL) {
    output_value =
VD_ACT_SERVO_GetAngleOutValueByRef(channelRef);
  }
  return output_value;
}


Std_PhyDataType
VD_ACT_SERVO_GetAngleOutValueByRef(VD_ACT_SERVO_Ch
annelType *channelRef)
{
  Std_PhyDataType output_value = 0;
  if (channelRef != NULL) {
    output_value = channelRef->curentAngle;
  }
  return output_value;
}



Std_ReturnType
VD_ACT_SERVO_SetAngleLimits(VD_ACT_SERVO_ChannelTy
```

```
Std_ReturnType
VD_ACT_SERVO_SetOutputValueByRef(VD_ACT_SERVO_Cha
nnelType *channelRef, Std_PhyDataType value)
{
  Std_ReturnType error = E_NOT_OK;
  if (channelRef != NULL) {

    if (value > channelRef->OUT_MAX)
      value = channelRef->OUT_MAX; //saturatie
    if (value < channelRef->OUT_MIN)
      value = channelRef->OUT_MIN; //saturatie

    channelRef->output_value = value;
    error = E_OK;
  }
  return error;
}


Std_PhyDataType
VD_ACT_SERVO_GetInputValue(Std_ChannelIdType
channelId)
{
  Std_PhyDataType input_value = 0;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if (channelRef != NULL) {
    input_value =
VD_ACT_SERVO_GetInputValueByRef(channelRef);
  }
  return input_value;
}


Std_PhyDataType
VD_ACT_SERVO_GetInputValueByRef(VD_ACT_SERVO_Chan
nelType *channelRef)
{
  Std_PhyDataType input_value = 0;
  if (channelRef != NULL) {
    input_value = channelRef->input_value;
  }
  return input_value;
}


Std_PhyDataType
VD_ACT_SERVO_GetOutputValue(Std_ChannelIdType
channelId)
{
  Std_PhyDataType output_value = 0;
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if (channelRef != NULL) {
    output_value =
VD_ACT_SERVO_GetOutputValueByRef(channelRef);
  }
  return output_value;
}


Std_PhyDataType
VD_ACT_SERVO_GetOutputValueByRef(VD_ACT_SERVO_Cha
nnelType *channelRef)
{
  Std_PhyDataType input_value =
VD_ACT_SERVO_GetInputValueByRef(channelRef);
  Std_PhyDataType output_value =
map_float(input_value, channelRef->IN_MIN,
channelRef->IN_MAX, channelRef->OUT_MIN,
channelRef->OUT_MAX);
  return output_value;
}



Std_ReturnType
VD_ACT_SERVO_SetInputLimits(VD_ACT_SERVO_ChannelT
```

```
pe *channelRef, Std_PhyDataType ANGLE_MIN,
Std_PhyDataType ANGLE_MAX)
{
  channelRef->ANGLE_MIN = ANGLE_MIN;
  channelRef->ANGLE_MAX = ANGLE_MAX;
  return E_OK;
}


Std_ReturnType
VD_ACT_SERVO_SetSpeedLimits(VD_ACT_SERVO_ChannelTy
pe *channelRef, Std_PhyDataType SPEED_MIN,
Std_PhyDataType SPEED_MAX)
{
  channelRef->SPEED_MIN = SPEED_MIN;
  channelRef->SPEED_MAX = SPEED_MAX;
  return E_OK;
}









Std_ReturnType
VD_ACT_SERVO_SetChannelRecurency(Std_ChannelIdType
channelId, int recurency)
{

  VD_ACT_SERVO_ChannelType *channelRef  =
VD_ACT_SERVO_GetChannelRef(channelId);

  if(channelRef == NULL)
    return E_NOT_OK;

  channelRef->servo_run_recurency = recurency;
  channelRef->accelerateRec =
(channelRef->acceleratePhy / (SECOND_MS /
recurency));
  return E_OK;
}

Std_ReturnType
VD_ACT_SERVO_SetGroupRecurency(Std_ChannelIdType
*srcIds, int recurency, uint8_t nr_of_channels)
{

  Std_ReturnType error = E_OK;

  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    error +=
VD_ACT_SERVO_SetChannelRecurency(srcId,
recurency);
  }
  return error;
}



#include "Arduino.h"

extern volatile int interruptCounter;
extern int CheckPoint_HERE;
Std_ReturnType VD_ACT_SERVO_ChannelRunnable(void*
parameters)
{
  Std_ReturnType error = E_OK;
  interruptCounter++;

  for (int channelId = 0; channelId <
VD_ACT_SERVO_CHANNEL_NR_OF; ++channelId) {
```

```
ype *channelRef, Std_PhyDataType IN_MIN,
Std_PhyDataType IN_MAX)
{
  channelRef->IN_MIN = IN_MIN;
  channelRef->IN_MAX = IN_MAX;
  return E_OK;
}


Std_ReturnType
VD_ACT_SERVO_SetOutputLimits(VD_ACT_SERVO_Channel
Type *channelRef, Std_PhyDataType OUT_MIN,
Std_PhyDataType OUT_MAX)
{
  channelRef->OUT_MIN = OUT_MIN;
  channelRef->OUT_MAX = OUT_MAX;
  return E_OK;
}


Std_ReturnType VD_ACT_SERVO_SetupComponent(void*
parameters)
{
  Std_ReturnType error = E_NOT_OK;

  return error;
}

Std_ReturnType
VD_ACT_SERVO_SetChannelRecurency(Std_ChannelIdTyp
e channelId, int recurency)
{

  VD_ACT_SERVO_ChannelType *channelRef  =
VD_ACT_SERVO_GetChannelRef(channelId);

  if(channelRef == NULL)
    return E_NOT_OK;

  channelRef ->run_recurency = recurency;



  return E_OK;
}

Std_ReturnType
VD_ACT_SERVO_SetGroupRecurency(Std_ChannelIdType
*srcIds, int recurency, uint8_t nr_of_channels)
{

  Std_ReturnType error = E_OK;

  for (size_t i = 0; i < nr_of_channels; i++)
  {
    Std_ChannelIdType srcId = srcIds[i];
    error +=
VD_ACT_SERVO_SetChannelRecurency(srcId,
recurency);
  }
  return error;
}



Std_ReturnType VD_ACT_SERVO_ChannelRunnable(void*
parameters)
{
  Std_ReturnType error = E_NOT_OK;

  return error;
}
```

```
    VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
    CheckPoint_HERE++;
    error +=
VD_ACT_SERVO_AngleProcess(channelRef);
  }

  return error;
}


Std_ReturnType
VD_ACT_SERVO_SpeedReset(Std_ChannelIdType
channelId) {
  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);
  if(channelRef == NULL)
    return E_NOT_OK;

  channelRef->currentSpeed = 0;

  return E_OK;
}



Std_ReturnType
VD_ACT_SERVO_AngleProcess(Std_ChannelIdType
channelId) {

  VD_ACT_SERVO_ChannelType *channelRef =
VD_ACT_SERVO_GetChannelRef(channelId);

  return VD_ACT_SERVO_AngleProcess(channelRef);
}


Std_ReturnType
VD_ACT_SERVO_AngleProcess(VD_ACT_SERVO_ChannelType
*channelRef) {

  // evaluate remaining distance
  channelRef->deltaAngle = channelRef->targetAngle
- channelRef->curentAngle;

  if(channelRef == NULL)
    return E_NOT_OK;

  //evaluate  direction
  if (channelRef->deltaAngle >
0.02)channelRef->servoDirection = SERVO_FWD;
  else if (channelRef->deltaAngle < -
0.02)channelRef->servoDirection = SERVO_BWD;
  else channelRef->servoDirection = SERVO_STP;

  // extract absolute value of the delta angle
  channelRef->deltaAngle =
abs(channelRef->deltaAngle);

  if (channelRef->servoDirection == SERVO_STP)
  { // reset if stop is reached
    channelRef->currentSpeed = 0;
    channelRef->accCurrentSpeed = 0;
    channelRef->decCurrentSpeed = 0;
    channelRef->servoNavState = SERVO_IDL;
  }
  else
  {
    // detect Accelerate decelerate indicators
    channelRef->accCurrentSpeed +=
channelRef->accelerateRec;
    channelRef->decCurrentSpeed = sqrt((double) (2
* channelRef->deltaAngle *
channelRef->acceleratePhy));

    // detect servo Navigate State
```

```
    if (min(channelRef->accCurrentSpeed,
channelRef->decCurrentSpeed)>=
channelRef->SPEED_MAX)
    {
      channelRef->servoNavState = SERVO_NAV;
      channelRef->currentSpeed =
channelRef->SPEED_MAX;
    }
    else if (channelRef->accCurrentSpeed <=
channelRef->decCurrentSpeed)
    {
      channelRef->servoNavState = SERVO_ACC;
      channelRef->currentSpeed +=
channelRef->accelerateRec;
    }
    else if (channelRef->accCurrentSpeed >
channelRef->decCurrentSpeed)
    {
      channelRef->servoNavState = SERVO_BRK;
      channelRef->currentSpeed -=
channelRef->accelerateRec;
    }

    channelRef->currentSpeed =
max(channelRef->currentSpeed,
channelRef->SPEED_MIN);

  }


  // angle evolution
  channelRef->curentAngle +=
channelRef->servoDirection *
(channelRef->currentSpeed / (SECOND_MS /
channelRef->servo_run_recurency)) ;

  // on forwarding
  if (channelRef->servoDirection == SERVO_FWD) {
    // saturate angle
    channelRef->curentAngle =
min(channelRef->curentAngle,
channelRef->targetAngle);
  }
  // on backwarding
  else if (channelRef->servoDirection ==
SERVO_BWD) {
    channelRef->curentAngle =
max(channelRef->curentAngle,
channelRef->targetAngle);
  // on Stopping
  } else {
  }

  return E_OK;
}
```

Anexa 7. Considerente de dezvoltare a aplicației de configurare și generare a componentelor pentru sistemele electronice distribuite

```
# module: configuration Generator
from datetime import date
from graphviz import Digraph

# importing os module
import os
import json


class AppGenerator:
  """Generate component configuration
Header."""

  def CfgHeadGen(self, jsonFile,
headerFile):
    """Generate component configuration
Header."""
    #
    with open(jsonFile, mode='r',
encoding='utf-8') as read_file:
      json_object = json.load(read_file)
      if "ProjectName" în json_object:
        application_name =
json_object["ProjectName"]
      else:
        application_name = "Project"

      if "Description" în json_object:
        appDescription =
json_object["Description"]
      else:
        appDescription = "ES Platform based
Project"

      file_gen = open(headerFile, mode='w',
encoding='utf-8')
      file_gen.write("/**\n")
      file_gen.write(" * @file " +
application_name + "_cfg_gen.h\n")
      file_gen.write(" * @author your name
(you@domain.com)\n")
      file_gen.write(" * @brief \n")
      file_gen.write(" * @version 0.1\n")
      file_gen.write(" * @date 2020-06-
12\n")
      file_gen.write(" * \n")
      file_gen.write(" * @copyright
Copyright (c) 2020\n")
      file_gen.write(" * \n")
      file_gen.write(" */\n")

      file_gen.write("#ifndef _" +
application_name.upper() + "_CONFIG_H_\n")
      file_gen.write("#define _" +
application_name.upper() + "_CONFIG_H_\n")
      file_gen.write("\n")
      file_gen.write("#include
\"./PLF/platform_config.h\"\n")
      file_gen.write("\n")
      linkComps = json_object["Components"]
      for comp în linkComps:

        # Ignore generation of the component
on ignore setting = true
        if "Ignore" în linkComps[comp]:
          if linkComps[comp]["Ignore"] ==
"true":
            continue

        # Override the name of the component
if defined by "Name"
        if "Name" în linkComps[comp]:
          componentName =
linkComps[comp]["Name"]
        else:
          componentName = str(comp)

        # Components Groups
        if "Groups" în linkComps[comp]:

          # no generation if no groups în
component identified
          if len(linkComps[comp]["Groups"])
== 0:
            continue

          # Override the default
configuration by defining
<Component>__CONFIG symbol
          file_gen.write("#define " +
componentName.upper() + "_CONFIG" + "\n")

          linkGroup =
linkComps[comp]["Groups"]
          for grp în linkGroup:

            # Ignore generation of the group
on ignore setting = true
            if "Ignore" în linkGroup[grp]:
              if linkGroup[grp]["Ignore"] ==
"true":
                continue

            # Override the name of the group
if defined by "Name"
            if "Name" în linkGroup[grp]:
              groupName =
linkGroup[grp]["Name"]
            else:
              groupName = str(grp)



            discipline = "SW"
            if "Discipline" în
linkGroup[grp]:
              discipline =
linkGroup[grp]["Discipline"]
```

```
            # do not generate channels
config în HEADER for non SW discupline
            if discipline != "SW" :
              continue

            # Channels
            if "Channels" în linkGroup[grp]:

              # no generation if no channels
în group identified
              if
len(linkGroup[grp]["Channels"]) == 0:
                continue

              file_gen.write("enum " +
groupName.upper() +
                     "_Cnl_IdType {")
              linkChannels =
linkGroup[grp]["Channels"]
              for cnl în linkChannels:
                file_gen.write(cnl + ", ")

file_gen.write(groupName.upper() +
                     "_CHANNEL_NR_OF};\n")

            # Defines
            if "Defines" în linkGroup[grp]:
              linkDefines =
linkGroup[grp]["Defines"]
              for cnl în linkDefines:
                file_gen.write("#define " +
cnl + " " +
                       str(linkDefines[cnl])
+ "\n")

        # Components Paths în project
        if "Path" în linkComps[comp]:
          linkPath = linkComps[comp]["Path"]
          file_gen.write("#include \"" +
str(linkPath) +
                   str(componentName) +
".h\"\n")

        file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
application_name + "_config(void);\n")

    file_gen.write("\n")
    file_gen.write("#endif // _" +
application_name.upper() + "_CONFIG_H_\n")
    file_gen.close()

  def CfgSrcGen(self, jsonFile, src_file):
    """Generate component configuration
Source file."""
    #
    with open(jsonFile, mode='r',
encoding='utf-8') as read_file:
      json_object = json.load(read_file)

      if "ProjectName" în json_object:
        application_name =
json_object["ProjectName"]
      else:
        application_name = "Project"

      if "Description" în json_object:
        appDescription =
json_object["Description"]
      else:
        appDescription = "ES Platform based
Project"

      linkComps = json_object["Components"]

      file_gen = open(src_file, mode='w',
encoding='utf-8')
      file_gen.write("/**\n")
      file_gen.write("* @file " +
application_name + "_cfg_gen.cpp\n")
      file_gen.write("* @author your name
(you@domain.com)\n")
      file_gen.write("* @brief \n")
      file_gen.write("* @version 0.1\n")
      file_gen.write("* @date 2020-06-12\n")
      file_gen.write("* \n")
      file_gen.write("* @copyright Copyright
(c) 2020\n")
      file_gen.write("* \n")
      file_gen.write("*/\n")
      file_gen.write("\n")
      file_gen.write("#include \"" +
application_name + "_cfg_gen.h\"\n")
      file_gen.write("\n")

# ////
      for comp în linkComps:

        # Ignore generation of the component
on ignore setting = true
        if "Ignore" în linkComps[comp]:
          if linkComps[comp]["Ignore"] ==
"true":
            continue

        # Override the name of the component
if defined by "Name"
        if "Name" în linkComps[comp]:
          componentName =
linkComps[comp]["Name"]
        else:
          componentName = str(comp)


        # Components Groups
        if "Groups" în linkComps[comp]:

          # no generation if no groups în
component identified
          if len(linkComps[comp]["Groups"])
== 0:
            continue

          linkGroup =
linkComps[comp]["Groups"]

          for grp în linkGroup:

            # Ignore generation of the group
on ignore setting = true
            if "Ignore" în linkGroup[grp]:
              if linkGroup[grp]["Ignore"] ==
"true":
                continue
```

```
            # Override the name of the group
if defined by "Name"
            if "Name" in linkGroup[grp]:
              groupName =
linkGroup[grp]["Name"]
            else:
              groupName = str(grp)

            # discipline = "SW"
            # if "Discipline" in
linkGroup[grp]:
            #   discipline =
linkGroup[grp]["Discipline"]
            # if discipline != "SW" :
            #   continue

            # Channels
            if "Channels" in linkGroup[grp]:

              # no generation if no channels
in group identified
              if
len(linkGroup[grp]["Channels"]) == 0:
                continue


file_gen.write("Std_ChannelIdType " +
groupName.upper() +
                    "_Group[")

file_gen.write(groupName.upper() +
                    "_CHANNEL_NR_OF] = {")
              linkChannels =
linkGroup[grp]["Channels"]
              cnl_str = ""
              for cnl in linkChannels:
                cnl_str = cnl_str + cnl + ",
"
              if cnl_str.endswith(', '):
                cnl_str = cnl_str[:-2]
              file_gen.write(cnl_str)
              file_gen.write("};\n")

              for cnl in linkChannels:

                # no generation if no
channels in group identified
                if len(linkChannels[cnl]) ==
0:
                  continue

                cnlName = str(cnl)
                lnkList = linkChannels[cnl]

file_gen.write("Std_ChannelIdType " +
groupName.upper() +
                "_" + cnlName.upper()
+"_Cnl["+ str(len(lnkList)) + "] = {")

                lnk_str = ""
                for lnk in lnkList:
                  lnk_str = lnk_str +
str(lnk) + ', '
                if lnk_str.endswith(', '):
                  lnk_str = lnk_str[:-2]

                file_gen.write(lnk_str)
                file_gen.write("};\n")
```

```
              file_gen.write("\n")

      # ////////////////////////////////

      file_gen.write("Std_ReturnType " +
application_name + "_config(void)\n")
      file_gen.write("{\n")
      file_gen.write("
Serial.begin(115200);\n")
      file_gen.write("  Serial.println(\"" +
appDescription + "\");\n")
      file_gen.write("  Std_ReturnType error
= E_OK;\n")


      for comp in linkComps:

        # Ignore generation of the component
on ignore setting = true
        if "Ignore" in linkComps[comp]:
          if linkComps[comp]["Ignore"] ==
"true":
            continue

        # Override the name of the component
if defined by "Name"
        if "Name" in linkComps[comp]:
          componentName =
linkComps[comp]["Name"]
        else:
          componentName = str(comp)



        # Components Groups
        if "Groups" in linkComps[comp]:

          # no generation if no groups in
component identified
          if len(linkComps[comp]["Groups"])
== 0:
            continue

          linkGroup =
linkComps[comp]["Groups"]

          for grp in linkGroup:

            # Ignore generation of the group
on ignore setting = true
            if "Ignore" in linkGroup[grp]:
              if linkGroup[grp]["Ignore"] ==
"true":
                continue

            # Override the name of the group
if defined by "Name"
            if "Name" in linkGroup[grp]:
              groupName =
linkGroup[grp]["Name"]
            else:
              groupName = str(grp)



            discipline = "SW"
            if "Discipline" in
linkGroup[grp]:
              discipline =
linkGroup[grp]["Discipline"]
```

```
            if discipline != "SW" :
              continue

            # Parent
            parentChannelStr = ""
            if "Parent" în linkGroup[grp]:
              parentChannelStr =
linkGroup[grp][
                "Parent"] + ", "

            # Channels
            if "Channels" în linkGroup[grp]:

              # no generation if no channels
în group identified
              if
len(linkGroup[grp]["Channels"]) == 0:
                continue

              file_gen.write("\n")
              linkChannels =
linkGroup[grp]["Channels"]

              # go through Cannels
              for cnl în linkChannels:

                # no generation if no links
în channel identified
                if len(linkChannels[cnl]) ==
0:
                  continue

                lnkList = linkChannels[cnl]

                # go through links
                for lnk în lnkList:



                  if (lnk != "null"):
                    file_gen.write("\
  error += " + componentName.upper() +
"_ChannelSetup(" +
                      parentChannelStr + cnl +
                      ", " + lnk +
                      ");\n")



            # Pull
            if "Pull" în linkGroup[grp]:

              file_gen.write("\n")
              linkChannels =
linkGroup[grp]["Channels"]

              for cnl în linkChannels:
                if (linkChannels[cnl] !=
"null"):
                  file_gen.write(
                    "\
  error += " + componentName.upper() +
"_SetPullMethod(" +
                      # parentChannelStr +
                      cnl + ", " +
                      linkGroup[grp]["Pull"]
+
                      ");\n")
```

```
            # Push
            if "Push" în linkGroup[grp]:

              file_gen.write("\n")
              linkChannels =
linkGroup[grp]["Channels"]

              for cnl în linkChannels:
                if (linkChannels[cnl] !=
"null"):
                  file_gen.write("
error += " + componentName.upper() +
"_SetPushMethod(" +
                    # parentChannelStr +
                    cnl + ", " +
                    linkGroup[grp]["Push"] +
");\n")

            file_gen.write("
Serial.print(\""+" Group: " + groupName + "
configured - Error : \");\n")
            file_gen.write("
Serial.println(error);\n")

        file_gen.write("\n")

    file_gen.write("  return error;\n")
    file_gen.write("}\n")
    file_gen.close()

  def PlfCompDemoHeadGen(self, comp_name,
header_file):
    """Generate platform component config
HEAD File """
    print("Start Generating platform
componet " + comp_name + " demo Header")

    file_gen = open(header_file, mode='w',
encoding='utf-8')

    file_gen.write("/**\n")
    file_gen.write(" * @file "+ comp_name
+"_demo.h\n")
    file_gen.write(" * @author " +
os.getlogin() + " (you@domain.com)\n")
    file_gen.write(" * @brief \n")
    file_gen.write(" * @version 0.1\n")
    file_gen.write(" * @date "+
date.today().strftime("%d-%b-%Y") +"\n")
    file_gen.write(" * \n")
    file_gen.write(" * @copyright Copyright
(c) 2020\n")
    file_gen.write(" * \n")
    file_gen.write(" */\n")
    file_gen.write("\n")

    file_gen.write("\n")
    file_gen.write("#ifndef _"+
comp_name.upper() +"_DEMO_H_\n")
    file_gen.write("#define _"+
comp_name.upper() +"_DEMO_H_\n")
    file_gen.write("\n")
    file_gen.write("void "+ comp_name
+"_demo_setup(void);\n")
    file_gen.write("void "+ comp_name
+"_demo_loop(void) ;\n")
    file_gen.write("\n")
```

```
    file_gen.write("\n")
    file_gen.write("#endif /* _"+
comp_name.upper() +"_DEMO_H_ */\n")
    file_gen.close()

  def PlfCompCfgHeadGen(self, comp_name,
header_file):
    """Generate platform component config
HEAD File """
    print("Start Generating platform
componet " + comp_name + " config Header")

    file_gen = open(header_file, mode='w',
encoding='utf-8')

    file_gen.write("/**\n")
    file_gen.write(" * @file "+ comp_name
+"_cfg.h\n")
    file_gen.write(" * @author " +
os.getlogin() + " (you@domain.com)\n")
    file_gen.write(" * @brief \n")
    file_gen.write(" * @version 0.1\n")
    file_gen.write(" * @date "+
date.today().strftime("%d-%b-%Y") +"\n")
    file_gen.write(" * \n")
    file_gen.write(" * @copyright Copyright
(c) 2020\n")
    file_gen.write(" * \n")
    file_gen.write(" */\n")
    file_gen.write("\n")
    file_gen.write("#ifndef _"+
comp_name.upper() +"_CFG_H_\n")
    file_gen.write("#define _"+
comp_name.upper() +"_CFG_H_\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("#include
\"./PLF/platform_config.h\"\n")
    file_gen.write("\n")
    file_gen.write("#ifndef "+
comp_name.upper() +"_CONFIG\n")
    file_gen.write("#define "+
comp_name.upper() +"_CONFIG\n")
    file_gen.write("//for demo application
purpose\n")
    file_gen.write("//shall be defined în
the platform\n")
    file_gen.write("enum "+
comp_name.upper() +"_Cnl_IdType {"+
comp_name.upper() +"_1,"+ comp_name.upper()
+"_2, "+ comp_name.upper() +"_3, "+
comp_name.upper() +"_CHANNEL_NR_OF};\n")
    file_gen.write("#endif\n")



    file_gen.write("\n")
    file_gen.write("#endif /* _"+
comp_name.upper() +"_CFG_H_ */\n")
    file_gen.close()



  def PlfCompHeadGen(self, comp_name,
header_file):
    """Generate platform component HEAD File
"""
    print("Start Generating platform
componet " + comp_name + " Header")
    #
    file_gen = open(header_file, mode='w',
encoding='utf-8')

    file_gen.write("/**\n")
    file_gen.write(" * @file " + comp_name
+".h\n")
    file_gen.write(" * @author " +
os.getlogin() + " (you@domain.com)\n")
    file_gen.write(" * @brief \n")
    file_gen.write(" * @version 0.1\n")
    file_gen.write(" * @date "+
date.today().strftime("%d-%b-%Y") +"\n")
    file_gen.write(" * \n")
    file_gen.write(" * @copyright Copyright
(c) 2020\n")
    file_gen.write(" * \n")
    file_gen.write(" */\n")
    file_gen.write("#ifndef _" +
comp_name.upper() +"_H_\n")
    file_gen.write("#define _" +
comp_name.upper() + "_H_\n")
    file_gen.write("\n")
    file_gen.write("#include
\"./PLF/platform_config.h\"\n")
    file_gen.write("\n")
    file_gen.write("#include \"" + comp_name
+ "_cfg.h\"\n")
    file_gen.write("\n")
    file_gen.write("#ifndef " +
comp_name.upper() + "_CONFIG\n")
    file_gen.write("enum " +
comp_name.upper() + "_Cnl_IdType {" +
comp_name.upper() + "_CHANNEL_NR_OF =
0 };\n")
    file_gen.write("#endif\n")
    file_gen.write("\n")
    file_gen.write("typedef struct " +
comp_name.upper() + "_ChannelType_t {\n")
    file_gen.write("\n")
    file_gen.write("Std_ChannelIdType
channelId = 0;\n")
    file_gen.write("\n")
    file_gen.write("Std_PhyDataType
phy_value = 0;\n")
    file_gen.write("Std_PhyDataType
raw_value = 0;\n")
    file_gen.write("\n")
    file_gen.write("Std_PhyDataType min_phy
= 0;\n")
    file_gen.write("Std_PhyDataType min_phy
= 100;\n")
    file_gen.write("\n")
    file_gen.write("Std_PhyDataType min_raw
= 0;\n")
    file_gen.write("Std_PhyDataType min_raw
= 100;\n")
    file_gen.write("\n")
    file_gen.write("Std_RawSetterType
SetChannelValue = NULL;\n")
    file_gen.write("Std_RawGetterType
GetChannelValue = NULL;\n")
    file_gen.write("\n")
    file_gen.write("int8_t
runnable_recurency = 10; \n")
    file_gen.write("\n")
    file_gen.write("\n")
```

```
    file_gen.write("} " + comp_name.upper()
+ "_ChannelType;\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("" + comp_name.upper() +
"_ChannelType* " + comp_name.upper() +
"_GetChannelRef(Std_ChannelIdType
channelId);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetupChannel(Std_ChannelIdType
servoChannelId, Std_ChannelIdType
channelId);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetupGroupChannels(Std_ChannelIdType
*srcIds, Std_ChannelIdType *targhetIds,
uint8_t nr_of_channels);\n")
    file_gen.write("\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetPushMethod(Std_ChannelIdType channelId,
Std_PhySetterType setterFunction);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetGroupPushMethod(Std_ChannelIdType
*srcIds, Std_PhySetterType setterFunction,
uint8_t nr_of_channels);\n")
    file_gen.write("\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetPullMethod(Std_ChannelIdType channelId,
Std_PhySetterType getterFunction);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetGroupPullMethod(Std_ChannelIdType
*srcIds, Std_PhySetterType getterFunction,
uint8_t nr_of_channels);\n")
    file_gen.write("\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetRawValue(Std_ChannelIdType channelId,
Std_RawDataType value);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetPhyValue(Std_ChannelIdType channelId,
Std_PhyDataType value);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetRawValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_PhyDataType angle);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetPhyValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_PhyDataType angle);\n")
    file_gen.write("\n")
    file_gen.write("Std_RawDataType " +
comp_name.upper() +
"_GetRawValue(Std_ChannelIdType
channelId);\n")
    file_gen.write("Std_PhyDataType " +
comp_name.upper() +
"_GetPhyValue(Std_ChannelIdType
channelId);\n")
    file_gen.write("Std_RawDataType " +
comp_name.upper() + "_GetRawValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef);\n")
    file_gen.write("Std_PhyDataType " +
comp_name.upper() + "_GetPhyValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef);\n")
    file_gen.write("\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetPhyLimits(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_PhyDataType phy_min,
Std_PhyDataType phy_max);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetRawLimits(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_RawDataType raw_min,
Std_RawDataType raw_max);\n")
    file_gen.write("\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetupComponent(void*
parameters);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_ChannelRunnable(void*
parameters);\n")
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetGroupRecurency(Std_ChannelIdType
*srcIds, int recurency, uint8_t
nr_of_channels);\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("#endif // _"+
comp_name.upper() + "_H_\n")
    file_gen.close()


  def PlfCompSrcGen(self, comp_name,
src_file):
    """Generate platform component SRC File
"""
    print("Start Generating platform
componet " + comp_name + " Source file")



    file_gen = open(src_file, mode='w',
encoding='utf-8')

    file_gen.write("/**\n")
    file_gen.write(" * @file " + comp_name
+".cpp\n")
    file_gen.write(" * @author " +
os.getlogin() + " (you@domain.com)\n")
    file_gen.write(" * @brief \n")
    file_gen.write(" * @version 0.1\n")
    file_gen.write(" * @date "+
date.today().strftime("%d-%b-%Y") +"\n")
    file_gen.write(" * \n")
    file_gen.write(" * @copyright Copyright
(c) 2020\n")
    file_gen.write(" * \n")
    file_gen.write(" */\n")
    file_gen.write("\n")
    file_gen.write("#include \"" + comp_name
+".h\"\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write(comp_name.upper() +
"_ChannelType " + comp_name.upper() +
```

```
"_Channels[" + comp_name.upper() +
"_CHANNEL_NR_OF];\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write(comp_name.upper() +
"_ChannelType* " + comp_name.upper() +
"_GetChannelRef(Std_ChannelIdType
channelId)\n")
    file_gen.write("  {\n")
    file_gen.write("  " + comp_name.upper()
+ "_ChannelType *channelRef = &" +
comp_name.upper() +
"_Channels[channelId];\n")
    file_gen.write("  return channelRef;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetupChannel(Std_ChannelIdType
servoChannelId, Std_ChannelIdType
channelId)\n")
    file_gen.write("  {\n")
    file_gen.write("    Std_ReturnType
error;\n")
    file_gen.write("    if (servoChannelId <
" + comp_name.upper() + "_CHANNEL_NR_OF)
{\n")
    file_gen.write("    " +
comp_name.upper() + "_ChannelType
*channelRef = " + comp_name.upper() +
"_GetChannelRef(servoChannelId);\n")
    file_gen.write("
            channelRef->pulseChannelId =
pulseChannelId;\n")
    file_gen.write("    error = E_OK;\n")
    file_gen.write("    } else {\n")
    file_gen.write("    error =
E_NOT_OK;\n")
    file_gen.write("    }\n")
    file_gen.write("    return error;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    # COMPONENT_SetupGroupChannels
    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetupGroupChannels(Std_ChannelIdType
*srcIds, Std_ChannelIdType *targhetIds,
uint8_t nr_of_channels)\n")
    file_gen.write("  {\n")
    file_gen.write("    Std_ReturnType error
= E_OK;\n")
    file_gen.write("    for (size_t i = 0; i
< nr_of_channels; i++)\n")
    file_gen.write("    {\n")
    file_gen.write("    Std_ChannelIdType
srcId = srcIds[i];\n")
    file_gen.write("    Std_ChannelIdType
targhetId = targhetIds[i];\n")
    file_gen.write("    error += " +
comp_name.upper() + "ChannelSetup(srcId,
targhetId);\n")
    file_gen.write("    }\n")
    file_gen.write("    return error;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetPushMethod(Std_ChannelIdType channelId,
Std_PhySetterType setterFunction)\n")
    file_gen.write("  {\n")
    file_gen.write("    Std_ReturnType
error;\n")
    file_gen.write("    if (channelId < " +
comp_name.upper() + "_CHANNEL_NR_OF) {\n")
    file_gen.write("    " +
comp_name.upper() + "_ChannelType
*channelRef = " + comp_name.upper() +
"_GetChannelRef(channelId);\n")
    file_gen.write("    channelRef->SetPulse
= SetPulse;\n")
    file_gen.write("    error = E_OK;\n")
    file_gen.write("    } else {\n")
    file_gen.write("    error =
E_NOT_OK;\n")
    file_gen.write("    }\n")
    file_gen.write("    return error;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetGroupPushMethod(Std_ChannelIdType
*srcIds, Std_PhySetterType setterFunction,
uint8_t nr_of_channels)\n")
    file_gen.write("  {\n")
    file_gen.write("Std_ReturnType error =
E_OK;\n")
    file_gen.write("\n")
    file_gen.write("for (size_t i = 0; i <
nr_of_channels; i++)\n")
    file_gen.write("{\n")
    file_gen.write("Std_ChannelIdType srcId
= srcIds[i];\n")
    file_gen.write("error += " +
comp_name.upper() + "_SetPushMethod(srcId,
deviceSetter);\n")
    file_gen.write("}\n")
    file_gen.write("return error;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetPullMethod(Std_ChannelIdType channelId,
Std_PhySetterType getterFunction)\n")
    file_gen.write("  {\n")
    file_gen.write("    Std_ReturnType
error;\n")
    file_gen.write("    if (channelId < " +
comp_name.upper() + "CHANNEL_NR_OF) {\n")
    file_gen.write("    " +
comp_name.upper() + "ChannelType *channelRef
= " + comp_name.upper() +
"_GetChannelRef(channelId);\n")
    file_gen.write("    channelRef->SetPulse
= SetPulse;\n")
    file_gen.write("    error = E_OK;\n")
```

```
    file_gen.write("    } else {\n")
    file_gen.write("    error =
E_NOT_OK;\n")
    file_gen.write("    }\n")
    file_gen.write("    return error;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetGroupPullMethod(Std_ChannelIdType
*srcIds, Std_PhySetterType getterFunction,
uint8_t nr_of_channels)\n")
    file_gen.write("  {\n")
    file_gen.write("Std_ReturnType error =
E_OK;\n")
    file_gen.write("\n")
    file_gen.write("for (size_t i = 0; i <
nr_of_channels; i++)\n")
    file_gen.write("{\n")
    file_gen.write("Std_ChannelIdType srcId
= srcIds[i];\n")
    file_gen.write("error += " +
comp_name.upper() + "_SetPullMethod(srcId,
deviceSetter);\n")
    file_gen.write("}\n")
    file_gen.write("return error;\n")
    file_gen.write("  }\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetRawValue(Std_ChannelIdType channelId,
Std_RawDataType value)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetPhyValue(Std_ChannelIdType channelId,
Std_PhyDataType value)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetRawValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_PhyDataType angle)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetPhyValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_PhyDataType angle)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_RawDataType " +
comp_name.upper() +
"_GetRawValue(Std_ChannelIdType
channelId)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_PhyDataType " +
comp_name.upper() +
"_GetPhyValue(Std_ChannelIdType
channelId)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_RawDataType " +
comp_name.upper() + "_GetRawValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_PhyDataType " +
comp_name.upper() + "_GetPhyValueByRef(" +
comp_name.upper() + "_ChannelType
*channelRef)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")
```

```
    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetPhyLimits(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_PhyDataType phy_min,
Std_PhyDataType phy_max)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetRawLimits(" +
comp_name.upper() + "_ChannelType
*channelRef, Std_RawDataType raw_min,
Std_RawDataType raw_max)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_SetupComponent(void*
parameters)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() + "_ChannelRunnable(void*
parameters)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.write("Std_ReturnType " +
comp_name.upper() +
"_SetGroupRecurency(Std_ChannelIdType
*srcIds, int recurency, uint8_t
nr_of_channels)\n")
    file_gen.write("  {\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("\n")
    file_gen.write("    return 0;\n")
    file_gen.write("  }\n")
    file_gen.write("\n")
    file_gen.write("\n")

    file_gen.close()
```

```
  def PlfCompDefGen(self, comp_name,
src_file):
    """Generate platform component DEF File
"""
    print("Start Generating platform
componet " + comp_name + " Definition")

    file_gen = open(src_file, mode='w',
encoding='utf-8')

    file_gen.write('{\n')
    file_gen.write('  "Components": {\n')
    file_gen.write('    "'+ comp_name +'":
{\n')
    file_gen.write('      "git":
"https://github.com/ML-ES-Platform/' +
comp_name +'.git",\n')
    file_gen.write('      "Path": "ESW/' +
comp_name +'/",\n')
    file_gen.write('      "Groups": {\n')
    file_gen.write('        "' + comp_name
+'": {\n')
    file_gen.write('          "NameSpace":
"' + comp_name +'",\n')
    file_gen.write('
"Multiplicity": "0-*",\n')
    file_gen.write('          "Push": [\n')
    file_gen.write('            "' +
comp_name.upper() +'_SetValue"\n')
    file_gen.write('          ],\n')
    file_gen.write('          "Pull": [\n')
    file_gen.write('            "' +
comp_name.upper() +'_GetValue"\n')
    file_gen.write('          ],\n')
    file_gen.write('          "Dependency":
[\n')
    file_gen.write('
"dd_pca9685",\n')
    file_gen.write('
"mcal_pwm"\n')
    file_gen.write('          ],\n')
    file_gen.write('          "Channels":
{\n')
    file_gen.write('
"Multiplicity": "1-*",\n')
    file_gen.write('            "NameSpace":
"' + comp_name.upper() +'"\n')
    file_gen.write('          },\n')
    file_gen.write('          "Defines":
{}\n')
    file_gen.write('        }\n')
    file_gen.write('      }\n')
    file_gen.write('    }\n')
    file_gen.write('  }\n')
    file_gen.write('}\n')

    file_gen.close()




  def DotGen(self, json_object, dotFile):
    """Generate project Linkage dot."""
    #
```

```
    dot = Digraph(comment='arm 6-dof demo')

    # with open(jsonFile, mode='r',
encoding='utf-8') as read_file:
    # json_object = json.load(read_file)

    application_name = ""

    if "ProjectName" in json_object:
      application_name =
json_object["ProjectName"]
    else:
      application_name = "Noname Project"

    # if "Components" not in json_object:
    #   return "null"

    layer_cluster = {
      "null" : []
    }


    linkComps = json_object["Components"]
    for comp in linkComps:

      # Ignore generation of the component
on ignore setting = true
      if "Ignore" in linkComps[comp]:
        if linkComps[comp]["Ignore"] ==
"true":
          continue

      cluster_name = "null"

      component_name = linkComps[comp]
      if "Domain" in linkComps[comp]:
        cluster_name =
linkComps[comp]["Domain"]

      # print(cluster_name)
      # print(comp)

      if cluster_name not in layer_cluster:
        layer_cluster[cluster_name] = []


layer_cluster[cluster_name].append(comp)



    for cluster_item in layer_cluster:

      cluster_name = str(cluster_item)

      with dot.subgraph(name="cluster_" +
cluster_name) as cl_node:
        if cluster_name != "null":
          cl_node.attr(color='blue')
          cl_node.node_attr['style'] =
'filled'
          cl_node.attr(label=cluster_name)
          #
cl_node.node(str("x"+cluster_name))

        for component_name in
layer_cluster[cluster_name]:
          if cluster_name == "null":
            cl_node = dot
            #
dot.node(str("x"+component_name))

          with
cl_node.subgraph(name="cluster_" +
str(component_name)) as comp_node:
            comp_node.attr(color='blue')
            comp_node.node_attr['style'] =
'filled'

comp_node.attr(label=component_name)
            #
comp_node.node(str("x"+component_name))


            # Components Groups
            if "Groups" in
linkComps[component_name]:
              if
len(linkComps[component_name]["Groups"]) ==
0:
                continue


              linkGroup =
linkComps[component_name]["Groups"]

              for grp in linkGroup:

                # Ignore generation of the
group on ignore setting = true
                if "Ignore" in
linkGroup[grp]:
                  if
linkGroup[grp]["Ignore"] == "true":
                    continue

                # Override the name of the
group if defined by "Name"
                if "Name" in linkGroup[grp]:
                  groupName =
linkGroup[grp]["Name"]
                else:
                  groupName = str(grp)


                # Channels
                if "Channels" in
linkGroup[grp]:

                  # no generation if no
channels in group identified
                  if
len(linkGroup[grp]["Channels"]) == 0:
                    continue

                  linkChannels =
linkGroup[grp]["Channels"]

                  for cnl in linkChannels:
                    # NOTE: the subgraph
name needs to begin with 'cluster' (all
lowercase)
                    #     so that Graphviz
recognizes it as a special cluster subgraph
                    # with
dot.subgraph(name="cluster_" + comp) as c:
```

```
                    #
comp_node.attr(color='blue')
                    #
comp_node.node_attr['style'] = 'filled'
                    #
comp_node.attr(label=comp)
                    comp_node.node(cnl)
                    # print(el)
                    src = cnl

                    linkList =
linkChannels[cnl]

                    for lnk in linkList:

                      dst = lnk
                      # if "Pull" in
linkGroup[grp]:
                      #   dst = el
                      #   src =
linkChannels[el]
                      dot.edge(src, dst)




    # print(layer_cluster)

    dot.render(dotFile, view=True)
    # dot.render('test-output/round-
table.gv', view=True)
    print(dot.source)

    # g = Digraph('G',
filename='cluster.gv')
    # g.view()

    print(" Diagram for "+ application_name
+ "generated ")

    return


    linkComps = json_object["Components"]
    for comp in linkComps:

      component_name = linkComps[comp]

      if (component_name != "null"):

        # Components Groups
        if "Groups" in linkComps[comp]:
          linkGroup =
linkComps[comp]["Groups"]
          for grp in linkGroup:
            groupName = str(linkGroup[grp])

            # Channels
            if "Channels" in linkGroup[grp]:
              linkChannels =
linkGroup[grp]["Channels"]
              for cnl in linkChannels:
                # NOTE: the subgraph name
needs to begin with 'cluster' (all
lowercase)
                #     so that Graphviz
recognizes it as a special cluster subgraph
                with
dot.subgraph(name="cluster_" + comp) as c:
                  c.attr(color='blue')
                  c.node_attr['style'] =
'filled'
                  c.attr(label=comp)
                  c.node(cnl)
                print(cnl)
                src = cnl
                dst = linkChannels[cnl]
                if linkChannels[cnl] !=
"null":
                  dot.edge(src, dst)
        # Components Groups
        if "Features" in linkComps[comp]:
          linkGroup =
linkComps[comp]["Features"]
          for ftr in linkGroup:
            groupName = str(linkGroup[ftr])

            # Channels
            if "Channels" in linkGroup[ftr]:
              linkChannels =
linkGroup[ftr]["Channels"]
              for cnl in linkChannels:
                # NOTE: the subgraph name
needs to begin with 'cluster' (all
lowercase)
                #     so that Graphviz
recognizes it as a special cluster subgraph
                with
dot.subgraph(name="cluster_" + comp) as c:
                  c.attr(color='blue')
                  c.node_attr['style'] =
'filled'
                  c.attr(label=comp)
                  c.node(ftr)
                print(cnl)
                src = ftr
                dst = cnl
                dot.edge(src, dst)

    dot.render(dotFile, view=True)
    # dot.render('test-output/round-
table.gv', view=True)
    print(dot.source)

    # g = Digraph('G',
filename='cluster.gv')
    # g.view()
```

Anexa 8. Considerente mecanice în proiectare de sistem

Conform conceptului general sistemele de tip IoT sunt sisteme de interconectare a lucrurilor sau al obiectelor, iar în acest context este important de definit care este obiectul, sistemele electronice distribuite reoferindu-se la totalitatea de tehnologii modele care implementează interacțiunile și gestionarea proceselor relevante aplicației în care este implicat obiectul sau lucrul. În cazul uscătoriei de fructe obiectul este reprezentat de construcția instalației de tip tunel prezentată în Fig. A8.3 iar funcționalitățile de control sunt proiectate pentru a asigura circulația aerului prin aceasta instalație pentru evacuarea controlată a umidității cum este prezentat în Fig. A8.1.

Sistemul de transport al materialului depinde de natura şi forma acestuia. Pentru fructe de formă circulară se folosesc tăvi metalice stivuite pe rafturile unor vagonete care se deplasează în cameră de la un capăt spre celălalt cu ajutorul unui dispozitiv mecanic. Cărucioarele încărcate se deplasează în tunel pe niște șine. Ele se încarcă în tunel peste un interval de timp concret pentru fiecare produs inițial.

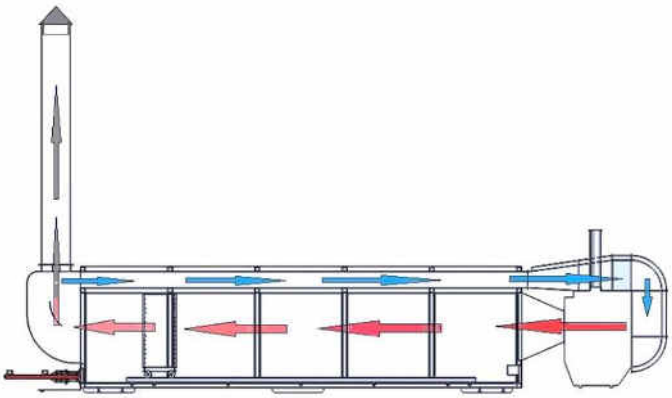


Fig. A8.1. Principiul de funcționare a camerei de uscare a fructelor

La un anumit interval de timp se introduc pe la un capăt al uscătorului vagonete cu material umed iar la capătul opus se scot același număr de vagonete cu material uscat. Agentul fierbinte, aerul, este antrenat de către ventilatorul ce este montat înainte de cazan. Acest aer este direcționat în camera de uscare, Fig. A8.1, preluând umiditatea de la produs, după care o parte din aer este evacuat în afară, iar restul este îndreptat înapoi în uscător prin canalul de reciclare din partea de sus a camerei de uscare. Astfel agentul de uscare efectuează un ciclu parțial închis, asigurând o uscare eficientă a produsului cere este predestinate uscării.

La baza automatizării uscătoriei de fructe este un modul electronic de control, bazat pe un sistem de calcul cu rol de măsurare, monitorizare a parametrilor necesari procesului de uscare și de a permite reprezentarea informațiilor pe panoul utilizatorului prin intermediul ecranului LCD, setarea parametrilor și dirijarea procesului de uscare. Pentru realizarea funcționalității uscătoriei de fructe e indispensabil un sistem de asigurare cu energie electrică, util pentru toate părțile componente și dirijarea în regim automat

a tuturor modulelor componente acționate electric. O schiță generala a componentelor ce compun sistemul este reprezentată în Fig. A8.2.

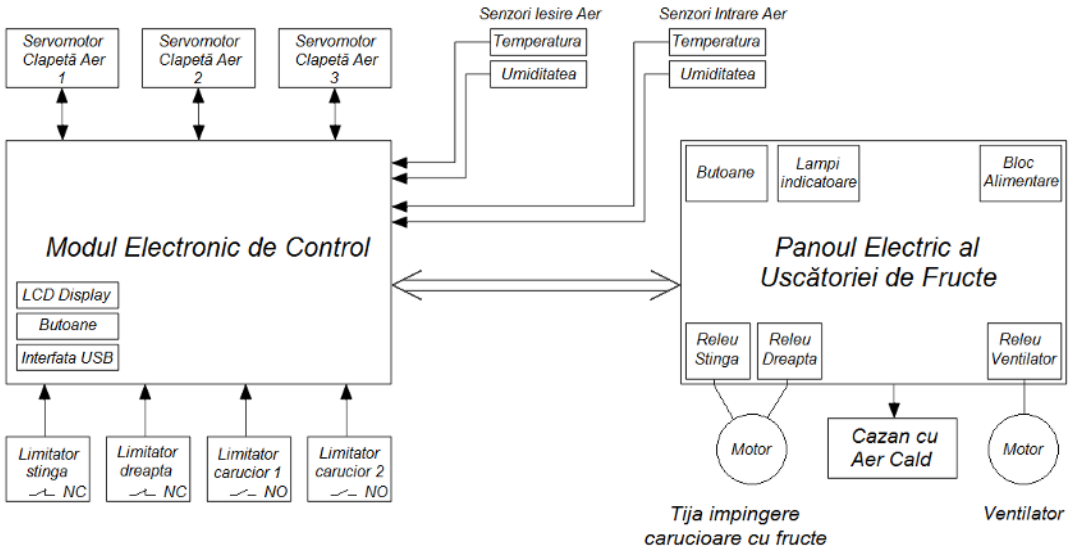


Fig. A8.2. Schița conceptuala a uscătoriei de fructe

Mai jos sunt prezentate funcționalitățile principale ale sistemului considerate la proiectarea sistemului de automatizare:

- Interfața cu utilizatorul - afișează parametrii procesului de uscare (temperatura, umiditatea aerului la intrarea și ieșirea din camera de uscare, unghiurile de deschidere a clapetelor de aer, durata de timp de la introducerea ultimului stelaj cu fructe în camera de uscare și începutul procesului de uscare, mesaje de eroare, alți parametri). Interfața permite introducerea anumitor caracteristici de setare a procesului de uscare prin intermediul butoanelor și a manipulatorului rotativ, realizând astfel interacțiunea cu operatorul;
- Panoului electric de dirijare –asigură alimentarea motorului ventilatorului principal, alimentarea motorului tijei de împingere a stelajului cu fructe în camera de uscare, pentru executarea mișcării acestuia în ambele direcții, asigurarea alimentării cazanului, asigurarea alimentării modulului electronic cu tensiune necesară, ș.a.;
- Controlul umidității – metodă de control a deschiderii clapetelor de aer la un anumit unghi, care reglează cantitatea de aer evacuat din camera de uscat fructe și si cantitatea de aer redirecționată spre recirculație;
- Controlul temperaturii – metodă de control a temperaturii în camera de uscare prin interacțiune a cu cazanul de aer cald prin activarea sau dezactivarea sursei;
- Monitorizarea parametrilor de lucru – metode de colectare a datelor de mediu din camera de uscare cum ar fi temperatura ,umiditatea și a fluxului de aer, cât și a informației despre starea elementelor

de acționare, clapeta sau tijele de manevrare a vagonetelor cu eventuala raportare valorilor curente și posibilelor erori;

- Introducere vagonete – Sistem de control a procesului de introducere a vagonetelor în camera de uscare prin acționarea tijei de împingere a stelajelor cu fructe;
- Colectare date – metoda de stocare date pentru utilizarea lor în studii statistici și optimizare a procesului de funcționare a camerei de uscare.
- Controlul sistemului – aplicație de control a sistemului ce implementează algoritmului general de funcționare a camerei de uscare integrând toate funcționalitățile sistemului și realizarea mecanismelor de monitorizare control diagnoze și protecții.

Servomotoarele clapetelor de aer, senzorii de temperatură și umiditate, precum și limitatoarele de cursă sunt conectate nemijlocit la Modulul Electronic de Control. Motorul tijei de împingere a stelajelor cu fructe în camera de uscare, motorul ventilatorului de aer, cazanul sunt conectate direct la Panoul Electric al Uscătoriei de Fructe. Interacțiunea dintre Modulul Electronic de Control și Panoul Electric se realizează prin intermediul semnalelor de Intrare / Ieșire, prin intermediul cărora se realizează comandarea acționării tijei de împingere a stelajelor cu fructe în interiorul camerei de uscare, aprinderea lămpilor indicatoare, sesizarea butoanelor de pe panoul electric, care sunt apăsate și verificarea stării releului ventilatorului (oprit / pornit).

Panoul electric conține blocul de alimentare, ce asigură alimentarea Modulului Electronic de Control cu tensiunea de 24 V curent continuu, necesară alimentării servomotoarelor clapetelor de aer, senzorilor de umiditate și sesizării stării limitatoarelor de cursă, releelor și butoanelor.

Construcția instalației de uscare tip tunel.

Construcția mecanica a instalației de uscare a fructelor de tip tunel este prezentată în Fig. A8.3 și este constituita din următoarele module mecanice: camera de uscare; canalul de reciclare a aerului; canalul de reîntoarcere parțială a aerului; canalul de redirecționare a aerului în cazan; țeavă de evacuare a aerului cu umiditate ridicată; ghidajele de tip cale ferată; vagonetele și tăvile de metal; mecanismul de deplasare a vagonetelor; cazanul pe combustibil solid;

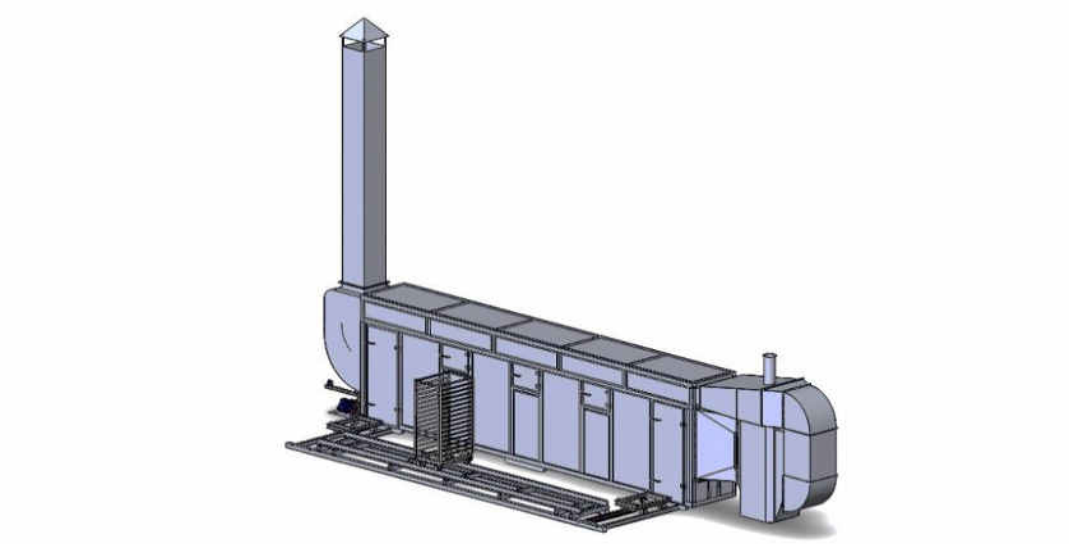


Fig. A8.3. Construcția mecanică de ansamblu a uscătoriei de tip tunel

Construcția mecanică a instalației implementează funcționalități mecanice ale sistemului la nivel de interacțiune cu mediul exterior, ele fiind în afara scopului acestei teze, însă în procesul de proiectare își aduce impactul prin impunerea de constrângeri de proiectare on domeniul ingineriei electrice și software prin intermediul parametrilor mecanici ale construcției.

Camera de uscare

Camera de uscare Fig. A8.4 este alcătuită: din profiluri metalice rectangulare, material termoizolator, foi de metal, cu dimensiunile de gabarit 7000*1200*1600mm. Este dotată cu doua uși, una pentru intrarea vagonetelor cu material umed, și cealaltă pentru extragerea acestor vagonete peste un interval de timp bine determinat cu materialul uscat. Pe aceeași parte sunt amplasate trei ferestre pentru observații și control asupra funcționării uscătorului, și a stării produsului. În interiorul camerei cărucioarele se deplasează în cameră de la un capăt spre celălalt cu ajutorul unui dispozitiv mecanic pe niște sine.

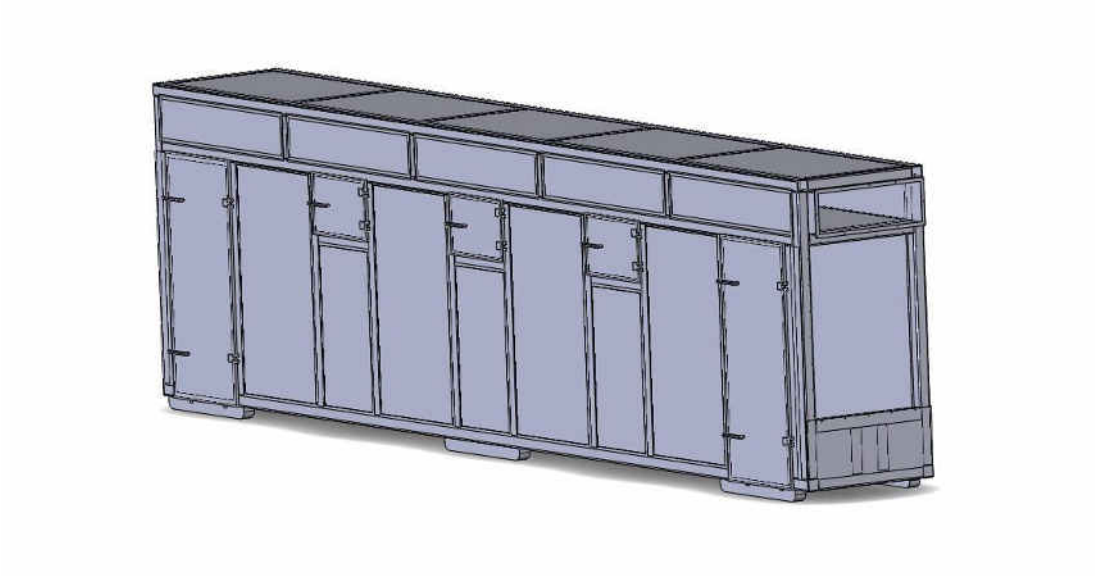


Fig. A8.4. Camera de uscare

Canalul de reciclare a aerului

Canalul de reciclare a aerului din Fig. A8.5 este situat deasupra camerei de uscare prin care are loc recircularea parțială a aerului uzat din camera de uscare cu dimensiunile de gabarit 7400*800*270mm.

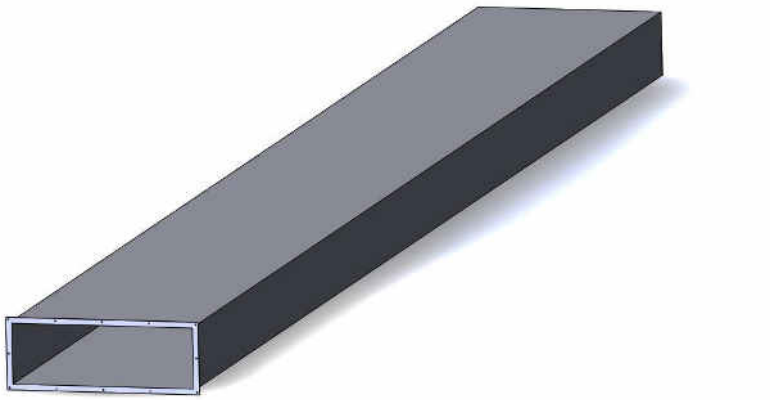


Fig. A8.5. Canalul de reciclare a aerului

Canalul de reîntoarcere parțială a aerului

Canalul de reîntoarcere parțială a aerului prezentat în Fig. A8.6 este cu dublă destinație: de redișionare a aerului, și de evacuare parțială a umidității, ce are loc datorită unei clapete ce funcționează automat, cu dimensiunile de gabarit 890*800*1670mm.

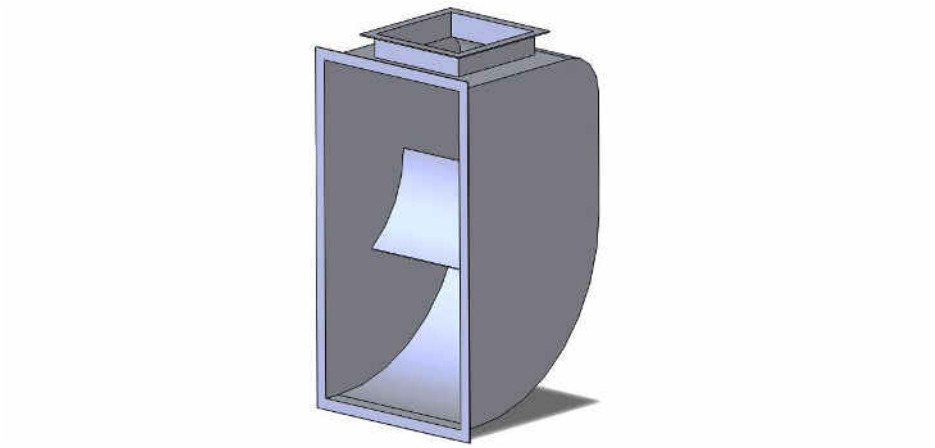


Fig. A8.6. Canalul de reîntoarcere parțială a aerului

Canalul de redirecționare a aerului în cazan

Canalul de redirecționare a aerului în cazan, Fig. A8.7, din zona cazanului servește pentru redirecționarea aerului la 180 de grade direcționându-l spre schimbătorul de căldură al cazanului. Acest canal este dotat cu o clapetă ce funcționează automat și este destinată pentru alimentare cu aer proaspăt. Având dimensiunile de gabarit 2320*950*2360mm

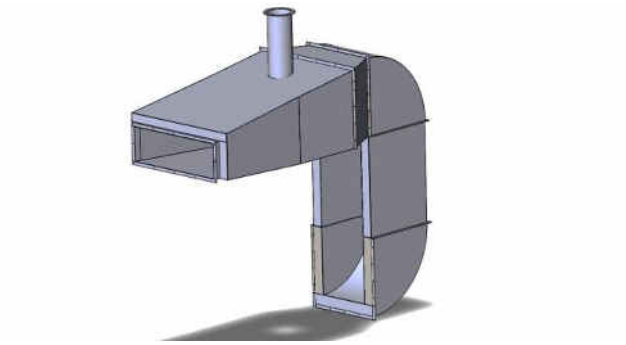


Fig. A8.7. Canalul de redirecționare a aerului în cazan

Ţeava de evacuare a aerului cu umeditate ridicată

Țeavă de evacuare a aerului cu umiditate ridicată, Fig. A8.8, are dimensiunile de gabarit 580*580*4270mm, și servește pentru evacuarea aerului umed.

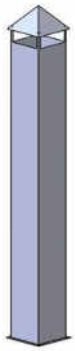


Fig. A8.8. Țeavă de evacuare a aerului cu umiditate

Ghidajele de tip cale ferată

Ghidajele de tip cale ferată Fig. A8.9 pentru deplasarea vagonetelor amplasate pe interior și exterior și au dimensiunile de gabarit 7160*2760*260mm.

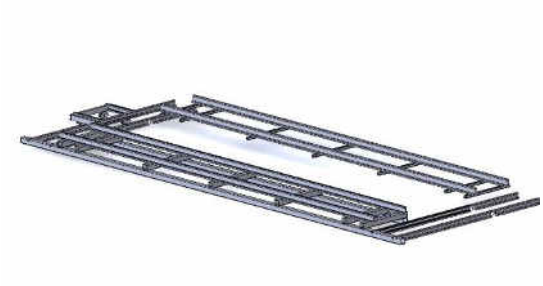


Fig. A8.9. Ghidajele de tip cale ferată

Vagonetele de transportare a fructelor

Vagonetele de transportare a fructelor prin uscătorie, Fig. A8.10, au dimensiunile de gabarit 560*750*1400mm.

Fig. A8.10. Vagonetele de transportare a fructelor

Mecanismul de deplasare a vagonetelor

Mecanismul de deplasare a vagonetelor din Fig. A8.11 este alcătuit din motor electric, reductor, arbore melcat, transmisii mecanice, cu dimensiunile de gabarit 1260*960*400mm

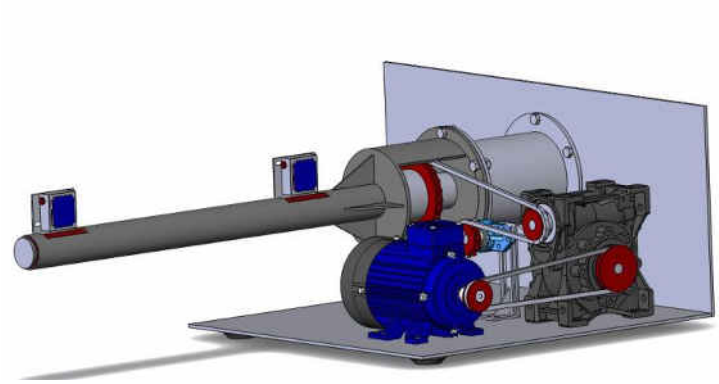


Fig. A8.11. Mecanismul de deplasare a vagonetelor

Cazanul pe combustibil solid

Cazanul pe combustibil solid Fig. A8.12, tip: EKOGREN EG-AIRMAX, cu dimensiunile de gabarit 1170*1150*1665mm

Fig. A8.12. Cazanul pe combustibil solid

Anexa 9. Considerente energetice în proiectare de sistem

Componentele de asigurare a funcționarii normale din punct de vedere al alimentarii cu energie electrice pot fi considerate auxiliare, cu implicații neesențiale în funcționalitățile de baza ale sistemelor, însă funcția acestora este una vitala, de asigurare cu energie electrica a sistemului. În cadrul proiectului uscătoriei de fructe, sistemul de asigurare cu energie electrica este redus la proiectarea distribuției energiei de la rețea către componentele instalației, fără a fi elaboră o oarecare interacțiune esențială de control proiectată la nivel software.

Repartizarea energiei electrice în încăpere

Uscătoria de Fructe este amplasată într-o încăpere, special construită pentru aceasta, cu suprafața de 12 x 14 m., anexa unei foste mici întreprinderi. Sarcină ce urma de rezolvat era proiectarea și realizarea repartizării tensiunilor necesare în încăpere, pentru asigurarea funcționalității Uscătoriei de Fructe. A fost proiectat și realizat un Panou Electric de Distribuției a Energiei Electrice în încăpere. Tensiunea alternativă în 3 faze este aplicată la acest panou direct de la contorul de energie electrică. Panoul trebuie să asigure alimentarea Uscătoriei de Fructe 1 și 2, iluminarea în încăpere, alimentarea unei prize de 3 faze, alimentarea instalațiilor ce urmează a fi procurate pe viitor – instalație de spălare a fructelor și a unui transportor. Schema bloc a Panoului Electric este prezentată în Fig. A2.1.

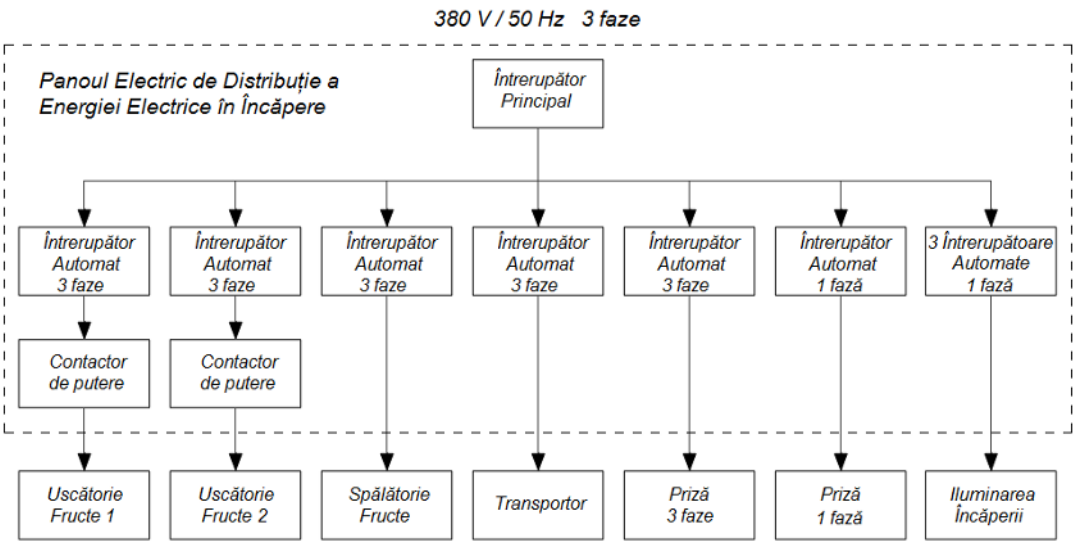


Fig. A2.1. Schema bloc a panoului Electric de Distribuție a Energiei Electrice în Încăpere

Întrerupătorul principal este amplasat în partea din față a Panoului și deconectează alimentarea tuturor sarcinilor. Este proiectat pentru un curent de 32 A, pentru a putea fi parcurs de un curent total destul de mare, curent sumar al tuturor instalațiilor consumatoare din încăperea uscătoriei. Fiecare instalație consumatoare este alimentată prin intermediul unor întrerupătoare automate, pentru a fi decuplată de la sursa de alimentare în caz de suprasarcină sau un eventual scurtcircuit. Uscătoriile de

Fructe sunt alimentate prin intermediul a câte un întrerupător automat de 16 A și câte un contactor de putere de 18 A. Pe partea anterioară a panoului sunt prevăzute butoane Start și Stop pentru conectarea acestora. Pe panou mai sunt două lămpi indicatoare de culoare verde, ce luminează atunci când începe alimentarea către uscătorii. La proiectarea panoului a fost prevăzută alimentarea instalațiilor, care urmează a fi procurate și instalate pe viitor – o instalație de spălare a fructelor, un transportor ce ar facilita procesul de pregătire a fructelor către uscare. O priză de 3 faze este preconizată pentru instalarea pe perete în apropierea panoului, pentru a putea alimenta o instalație portabilă cu alimentare trifazată, de exemplu, un aparat de sudat sau un compresor de aer sau orice alt echipament ce se alimentează de la 380 V trifazat. Concomitent, este planificată alimentarea unei prize de 220 V curent alternativ prin intermediul unui întrerupător automat, preconizată de-a fi fixată pe perete în apropierea panoului, cu scop de alimentare a unei sarcini monofazate. Iluminarea spațiului are stabilit 3 întrerupătoare automate în panou, care pot fi grupate pe zone de lucru cu posibilitatea deconectării în succesiune. În caz de un eventual scurtcircuit în rețeaua electrică a sistemului de iluminat, acesta nu v-a conteni în totalitate, doar va întrerupe alimentarea zonei afectate. Panoul Electric este dotat în partea inferioară cu contacte specializate, care ușurează procesul de montare a acestuia în încăpere și conectarea consumatorilor la panou. Imaginile panoului și elementelor componente sunt prezentate în Fig. A2.2.

Fig. A2.2. Panoul Electric de Distribuție a Energiei Electrice și elementele componente

Panoul electric al Uscătoriei de Fructe

Panoul Electric al Uscătoriei de Fructe are funcția de asigurare cu tensiune de alimentare a Modulul Electronic de Control, motoarele ventilatorului, tijei de împingere a stelajelor cu fructe în camera de uscare, precum și de a permite utilizatorului să pornească și să oprească ventilatorul, ciclul tijei. Partea anterioară a panoului este dotată cu butoane, cu lămpi indicatoare ce semnalează pornirea motorului ventilatorului, care asigură circulația aerului în camera de uscare, starea limitatoarelor ce indică prezența cărucioarelor la intrare și ieșire din camera de uscare, precum și apariția unor mesaje de eroare sau

atenționare pe parcursul procesului de uscare. Schema bloc a Panoului Electric al Uscătoriei de Fructe este prezentată în Fig. A2.3.

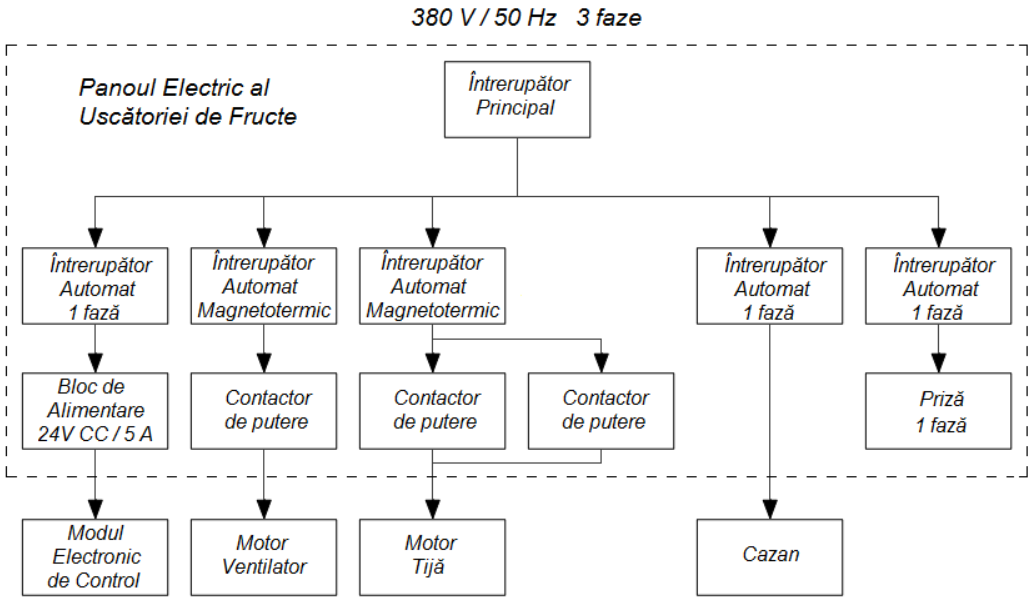


Fig. A2.3. Schema bloc a Panoului Electric al Uscătorului de Fructe

Blocul de Alimentare asigură tensiunea de 24 V curent continuu pentru alimentarea Modulului Electronic de Control, servomotoarelor clapetelor de aer, senzorilor de umiditate și sesizării stărilor limitatoarelor de cursă, releelor, butoanelor prin intermediul semnalelor de Intrare și Ieșire. Acesta este unul din principiile de comutație, ce asigură un randament crescut la o putere mare de ieșire și o stabilitate mai mare a tensiunii de ieșire la fluctuațiile tensiunii în rețea. Alimentarea acestuia se face prin intermediul unui întrerupător automat monofazat, cu destinație de protecție în caz de suprasolicitare.

Alimentarea motorului ventilatorului se face prin intermediului unui întrerupător automat magneto-termic, ce susține o protecție sporită motorului la depășirea curentului de consum. Curentul limită de declanșare a întrerupătorului dat poate fi reglat în diapazonul 4 – 6 A, ceea ce este potrivit motorului ventilatorului de 2,2 kW cu un curent de consum nominal de 4,7 A. Pornirea și oprirea motorului ventilatorului se face prin intermediul butoanelor Start și Stop din partea din față a panoului, care acționează un contactor de putere. Lampa indicatoare de culoare verde a panoului electric indică operatorului că ventilatorul este pus în mișcare. O grupă de contacte adăugătoare ale contactorului formează un semnal pentru indicarea Modulului Electronic de Control, indicativ al funcționalității ventilatorului.

Alimentarea motorului, care acționează tija de împingere a stelajelor cu fructe, se face de asemenea prin intermediul unui întrerupător automat magneto-termic și unul din două contactoare de putere. Unul folosește la rotirea motorului într-o direcție și al doilea în direcția opusă, pentru realizarea mișcărilor spre dreapta și stânga a tijei. Alimentarea bobinelor contactoarelor este trecută printr-o grupă de contacte de

tip Normal Închis ale contactoarelor direcției opuse, spre evitarea situațiilor de scurtcircuit a două faze. Contactoarele sunt comandate de semnalele de ieșire ale Modulului Electronic de Control corespunzător cu fazele ciclului de împingere a stelajelor cu fructe în camera de uscare.

Pornirea ciclului de acționare a tijei este comandată de butoanele respective din partea anterioară a panoului, reprezentând semnale de intrare pentru Modulul Electronic de Control, iar lămpile indicatoare ale prezenței cărucioarelor (culoarea galbenă) și apariției mesajelor de eroare și atenționare (culoarea roșie) reprezintă semnale de ieșire. Alimentarea Cazanului Uscătoriei de Fructe se face la fel prin intermediul Panoului Electric și de asemenea este separată printr-un întrerupător automat monofazat. În interiorul Panoului Electric al Uscătoriei de Fructe este încadrată o priză monofazată, separată de un întrerupător automat monofazat, care alimentează echipamentele specificate, necesare intervenției asupra automatizării uscătoriei. Panoul Electric este dotat în partea inferioară cu contacte speciale, cu funcția de ușurare a procesului de montare a acestuia, conectarea motoarelor și semnalelor către el. Imaginile panoului și elementelor componente sunt prezentate în Fig. A2.4.

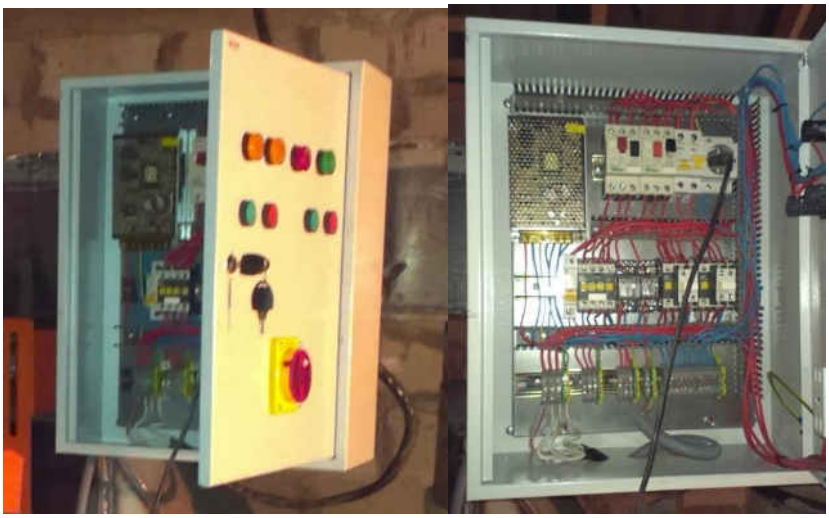


Fig. A2.4. Panoul Electric al Uscătorului de Fructe și elementele componente

Anexa 10. Certificate și dovezi de implementare

APROB

Administrator S.R.L. „Viomarix-Plus"

Guțan Mariana

ACT

de implementare a rezultatelor tezei de doctor în tehnică
„Modelarea sistemelor electronice distribuite cu o evoluție
autonomă si dinamică a configurației"
a dlui Bragarenco Andrei

Membrii consiliului tehnic ai S.R.L. „Viomarix-Plus" au încheiat prezentul act care confirmă că în baza rezultatelor tezei de doctor în tehnică „Modelarea sistemelor electronice distribuite cu o evoluție autonomă si dinamică a configurației" elaborată de dl. Bragarenco Andrei, firma S.R.L. „Viomarix-Plus" a implementat în producere o instalație de uscare în bază de pelete pentru fructe și legume cu capacitatea de 1,5 tone materie primă.

În rezultatul implementării, a fost instalat un sistem de automatizare la instalația de uscare sus numită, care a contribuit la reducerea consumului de energie cu 15%, constituind 51kW/h.

Director S.R.L. „Viomarix-Plus"

Guțan Marcel

A.O. Asociatia Artelor Alternative
Chisinau, Republica Moldova

Chisinau, 10 Aprilie, 2023

Scrisoare de confirmare

Prin prezenta confirm ca în perioada anilor 2013-2015, Asociația ArtWatt a realizat conceptul artistic a pavilionului republicii Moldova la expoziția mondială EXPO Milano -2015. Conceptul pavilionului a constat din trei lucrări cheie - printre care proiectul "Floarea Solară". Instalația dată a fost realizată în colaborare cu Andrei Bragarenco, unde el a implementat în baza rezultatelor tezei sale de doctor în informatică cu tema „Modelarea sistemelor electronice distribuite cu generarea automată a configurației" sistemul de iluminare a clădirii cu lumina ambiantă prin intermediul a patru oglinzi situate la fiecare colț a pavilionului, pentru urmărirea mișcării soarelui și reflectarea luminii către o țintă specifică, situată în centru.

Conceptul pavilionului a fost nominalizat ca cel mai bun proiect în spațiul public la concursul Top-Idea Awards 2015 in orașul Shenzhen, China.

Asociația Artelor Alternative «ARTWATT»

Pavel Braila

Președinte „ART WATT"

A P R O B

Director S.R.L. „Arobs Software"

Mihai Gorgos

ACT

de implementare a rezultatelor tezei de doctor în informatică
„Modelarea sistemelor electronice distribuite cu generarea
automată a configurației"
a dlui Bragarenco Andrei

Membrii consiliului tehnic ai S.R.L. „Arobs Software" au încheiat prezentul act care confirmă că în baza rezultatelor tezei de doctor în informatică „Modelarea sistemelor electronice distribuite cu generarea automată a configurației" elaborată de dl. Bragarenco Andrei, firma S.R.L. „Arobs Software" a implementat un sistem de monitorizare a mediului înconjurător prin intermediul amplasării dispozitivelor electronice în diferite puncte carteziene de interes dotate cu senzori în diverși parametri fizici, cum ar fi temperatură, umiditate, luminozitate, CO, mișcare, nivel zgomot.

În rezultatul implementării, a fost produs un prototip permite construirea hărților de mediu în diverși parametri fizici în baza datelor colectate din rețeaua sistemului electronic distribuit, prezentat la expoziția Embedded World 2019 – Nuernberg, Germania..

Director S.R.L. „Arobs Software"

Mihai Gorgos _____

Extras din procesul - verbal nr. 1

al Comisiei de concurs

în cadrul Concursului Cursurilor digitale UTM, ediția toamnă 2021, a.u. 2021/2022

30 noiembrie 2021

Prezenți

Vladislav Reşitca – prorector pentru studii, Nicolae Secrieru – conf. univ. dr, FET, Petru Todos – conf. univ. dr, FEIE, Natalia Burlacu– conf. univ. dr, FCIM, Raisa Druță – conf. univ. dr, FTA, Elena Sidorenco. – conf. univ. dr, FCGC, Tatiana Munteanu - conf. univ. dr, FIEB, Bernaz L. – secretarul comisiei, DMAAC.

Ordinea de zi:

1. Totalizarea rezultatelor Concursului cursurilor digitale, ediția toamnă 2021

S-a hotărât:

Locul I se acordă cursului Internetul Lucrurilor (IoT), autor Bragarenco Andrei, FCIM - 45,38 puncte.

Secretar al Comisiei de concurs _____ L. Bernaz

DECLARAȚIA PRIVIND ASUMAREA RĂSPUNDERII

Subsemnatul, declar pe proprie răspundere că materialele prezentate în teza de doctorat sunt rezultatul propriilor cercetări și realizări științifice, în caz contrar urmând să suport consecințele, în conformitate cu legislația în vigoare.

Bragarenco Andrei

Semnătura 

Data: 04.06.2022

CURRICULUM VITAE

Date personale

Nume și Prenume: BRAGARENCO ANDREI
Data și locul nașterii: 17.07.1979,
s. Volintiri, r. Ștefan Vodă, Republica Moldova;
Situația familială: căsătorit, trei copii
Studii: Universitatea Tehnică a Moldovei, Chișinău (1996-2002); Universitatea Tehnică a Moldovei (2003-2004); Universitatea Tehnică Gh. Asachi Iași (2006-2007).

Formarea profesională:

2002 – diplomă de licență, specializarea „Microelectronica", facultatea Calculatoare Informatică și Microelectronică, Universitatea Tehnică a Moldovei.
2004 – diplomă de masterat, specializarea „Sisteme Informaționale, Software și management", Universitatea Tehnică a Moldovei.
2007 – diplomă de masterat, specializarea „Convertoare Electronice de Putere", Universitatea Tehnică Gh. Asachi Iași.
2004 – 2007 – studii de doctorat, specialitatea 05.27.01 „Electronica corpului solid, microelectronică, nanoelectronică", Universitatea Tehnică a Moldovei.
2018 – 2023 – pretendent doctorat, specialitatea 122.03 „Modelare, metode matematice, produse program", Universitatea Tehnică a Moldovei.

Activitatea profesională:

2003-2005 – Cercetător științific, catedra MDS, UTM.
2004 – prezent – Lector asistent (superior) departamentul MIB, UTM.
2007 – Inginer proiectare și verificare Circuite Integrate, *AsicART srl* Iași, Romania.
2007 – 2014 – Inginer tehnic principal, *Micrologic Design Automation srl*, Chișinău, r. Moldova.
2011 – Inginer verificare Circuite Integrate Digitale, *Silicon Service srl* Iași, Romania.
2011 – prezent – Director/PM, *A.O. Clubul Ingineresc Micro Lab*, Chișinău, r. Moldova.
2015 – 2020 – Team Lead, SW Safety Engineer, *AROBS Software srl*, Chișinău, r. Moldova.
2021 – prezent – Project Engineer for Safety, *Continental AG*, Iași, Romania.

Cursuri universitare elaborate și ținute: Structuri de Date și Algoritmi, Microprocesoare și Interfețe, Proiectarea Microsistemelor, Programarea în Electronica, Sisteme Electronice Programabile, Limbaje de Descriere Hardware, Proiectarea și Verificarea Circuitelor Digitale, Proces de Dezvoltare a Sistemelor Incorporate, Sisteme Electronice Incorporate, Embedded Systems(en), Internetul Lucrurilor (IoT), Aplicatii ale Sistemelor Robotice, Problem Based Learning (PBL).

Cursuri extra-curriculare elaborate și ținute: Școala de vară – Sisteme incorporate, Autonomous Driving Bootcamp, Robot Factory – Mechatronix Symphony, IoT Bootcamp (+StayHome editie covid19), IT Driven Hardware – PCB design Bootcamp, Agro IT Academy, Engineering Internship (PBL)

Domeniile de activitate științifică: Proiectarea sistemelor Incorporate, Modelarea sistemelor de tip Internetul Lucrurilor, Proiectarea echipamentelor Industriale și mecatronica, Modelarea și proiectarea sistemelor Automotive. Modelarea conceptelor de siguranța în Automotive.

Participări în proiecte internaționale și naționale:
Proiect de inovare și transfer tehnologic „14.824.03.187T Elaborarea și implementarea uscătoriei în bază de pelete pentru fructe și legume", perioada de desfășurare 2014.01.02 - 2014.12.31

Cunoașterea limbilor: Română, Engleză, Rusă.

Date de contact Tel. mob. (373 79) 99-32-55 e-mail: andrei.bragarenco@microlab.utm.md