THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Cotutelle internationale: ITMO University

Présentée et soutenue par

Dmitry LEVSHUN

Le 15 décembre 2021

MODELES, ALGORITHMES ET METHODOLOGIE POUR LA CONCEPTION DE SYSTEMES DE SECURITE PHYSIQUE BASES SUR DES MICROCONTROLEURS PROTEGES DES ATTAQUES CYBER-PHYSIQUES

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Yannick CHEVALIER et Igor KOTENKO

Jury

M. Andrei PETROVSKI, Rapporteur
M. Vladimir OLESHCHUK, Rapporteur
M. Costin BADICA, Rapporteur
M. Vasily DESNITSKY, Examinateur
M. Roland RIEKE, Examinateur
Mme Ileana OBER, Examinatrice
M. Yannick CHEVALIER, Directeur de thèse
M. Igor KOTENKO, Co-directeur de thèse

v. 1.0.3

MODELS, ALGORITHMS AND METHODOLOGY FOR DESIGN OF MICROCONTROLLER-BASED PHYSICAL SECURITY SYSTEMS PROTECTED FROM CYBER-PHYSICAL ATTACKS

Thesis submitted by

Dmitry Levshun

under the guidance of

Prof. Igor Kotenko, ITMO University, Russia
Yannick Chevalier, University Paul Sabatier, France

under the consultation of

Andrey Chechulin, SPC RAS, Russia

in partial fulfillment of the requirements
for the award of the degree of

Doctor of Philosophy in Computer Science

ITMO UNIVERSITY

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

2021

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisors Igor Kotenko and Yannick Chevalier for giving me freedom in research, for their patience, motivation, enthusiasm and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

Besides my supervisors, I would like to thank Andrey Chechulin for constant help and support on the way in science, for making me interested in the design of secure microcontroller-based systems, for his insightful comments and difficult questions.

My sincere thanks also goes to Sergei Soloviev and Martin Strecker for their support during my stay in France as well as their assistance in the organization of joint supervision of my thesis.

I want to thank my colleagues from the Laboratory of Computer Security Problems: Igor Saenko, Vasily Desnitsky, Elena Fedorchenko, Maxim Kolomeec and Diana Gaifulina for the stimulating discussions, sleepless nights we were working together before deadlines and all the fun we have had since my join.

I would also like to express my gratitude to the staff of ITMO University from the Department of International Postgraduate and Doctoral Studies, Faculty of Secure Information Technologies and Department for Work with Dissertation Councils for their continued support during my studies.

A similar thanks goes to the staff of University of Paul Sabatier from the Ecole Doctorale Mathématique Informatique Télécommunications de Toulouse and Institut de Recherche en Informatique de Toulouse.

This work would have also been impossible without support of the St. Petersburg Federal Research Center of the Russian Academy of Sciences, which provided research equipment and approved long-term missions to France.

I also want to thank the Russian Foundation for Basic Research that supported this work based on the 19-37-90082 project. This support helped me to focus on the research presented in this work.

Last but not least, I want to thank my family and friends that were always here for me and supported me even if they do not fully understand what exactly I am doing :)

The latest versions of this document and source code of the developed application are available on https://github.com/levshun/PhD-mcbpss_design.

Contents

Introduction

Relevance of the topic of research. Microcontroller-based systems now are an integral part of any sphere of our vital activity, that is why the importance of ensuring their security is critical. The consequences of failure of such systems, including those associated with the activities of intruders, include both financial and reputational damage as well as a threat to human life and health. One of the possible attack vectors is the exploitation of vulnerabilities, the presence of which in microcontroller-based systems is due to various factors.

Vulnerabilities that occur due to errors at the design stage are the most dangerous, because after the system implementation, its improvement can be a difficult task. Especially when the improvement implies changes in the hardware or software components of individual devices, manufacturers of which no longer exist. The prevalence of such vulnerabilities is related to the fact that usually systems are designed without the participation of security experts using unsecure data transfer protocols and untested code.

For example, according to the SonicWall report, microcontroller-based devices malware attacks jumped 215.7% to 32.7 million in 2018 (up from 10.3 million in 2017). In 2019, the attacks continued but showed a more moderate increase of 5%, according to their 2020 Cyber Threat Report. And according to the Palo Alto Networks 2020 Unit 42 Threat Report “98% of all device traffic is unencrypted, exposing personal and confidential data on the network”.

Solving this problem is an important task, that is why various design techniques have been developed and embedded into practice. Some of them are focused on software, some on hardware, and some on highly specialized areas of the application. The key issue of such solutions is in focusing on certain aspects of the security, ensuring their inapplicability for providing the security of microcontroller-based systems in general.

For example, techniques for software do not take into account that the functionality of individual components of microcontroller-based systems is determined not only by software, but also by hardware. Moreover, the relationship between hardware and software elements can be quite strong, which leads to additional restrictions that significantly affect the process of their design and development.

An important drawback of the techniques for hardware and software-hardware is that the designed microcontroller-based device is viewed in isolation from the system. It means that not all security aspects would be taken into account and the security of the system as a whole will not be ensured. Also, there are extensions of these techniques that are aimed at ensuring the security of the devices and network

between them. The drawback is that such techniques provide secure connection between designed and external systems only from the designed system side, which can lead to security issues in complex multi-level systems.

In addition, in the area of techniques for links between devices only solutions applicable within a specific platform and architecture are widely used. Such solutions are aimed at adapting secure Internet protocols for their application as part of the interaction between microcontroller-based devices. The need for such adaptations is associated with the limited computing power of such devices, the size of the payload available for transmission in the data channel, and the ability to store relatively small amounts of data on them.

Commercial solutions from Google, ARM, Kaspersky, Microsoft, Siemens and Intel are not applicable if the microcontroller-based system already contains devices whose hardware cannot be changed or the design requirements contain restrictions that do not allow the use of devices suitable for these requirements. These solutions also do not take into account the optimization process of the designed system due to limitations like computational complexity, energy efficiency, size and price. It means that the resulting system may not be reasonable for a developed use case because of no trade-off between resources and security level.

In addition, there are many solutions in which the security of the system is not considered or is not the main task. At the same time, integration of the standalone solutions within a single approach is a difficult task due to their incompatibility. This is because each design technique is based on its own model of the system, presented in an internal format. That is why it is difficult or even impossible to transform one particular model into another without losses of significant data.

It means that a general approach for solving the issue of secure microcontroller-based systems design is not done yet. Therefore, the thesis research is aimed at developing the original model-methodological apparatus for the design of microcontroller-based physical security systems. Among all possible microcontroller-based systems, in this work only physical security systems were chosen as an area of the application, because in such systems during the design process it is required to ensure not only the functionality of the system, but also to ensure its security against cyber-physical attacks.

The degree of elaboration of the topic. The design of secure microcontroller-based systems is the subject of the works of such scientists as Abdelwahed S., Achiche S., Al-Muhtadi J., Ardeshiricham A., Balasubramaniyan S., Blanchet B., Bradley D., Bresolin D., Broy M., Bu L., Buonopane F., Cai S., Cai Y., Chechulin A., Chen G., Cremers C., Derhab A., Desnitsky V., Dong X., Eynard B., Faily S., Fukazawa Y., Geretti L., Gurjanov A., Han Z., Hannis M., Hao Q., He B., Hehenberger P., Hu F., Hu W., Huang C., Huang J., Iannucci S., Kaiya H., Karpovsky

M., Kastner R., Khaitan S., King J., Kobashi T., Kotenko I., Leonard L., Li X., Lin Z., Liu C., Lu J., Lu Y., Lü J., Ma R., Marxen J., McCalley J., Montemaggio A., Myers A., Nechaev V., Nuzzo P., Okubo T., Patalano S., Patil Y., Penas O., Plateaux R., Ramaswamy S., Saleem K., Sangiovanni-Vincentelli A., Shahzad B., Srinivasan S., Subathra B., Suh G., Tomiyama T., Vain J., Vasilakos A., Villa T., Vogel-Heuser B., Wang B., Wang H., Wang Y., Wang Z., Washizaki H., Xia X., Xiong N., Xu X., Yang W., Yoshioka N., Yu S., Zakoldaev D., Zhang D., Zhang T., Zharinov I., Zhong S., Zhou X., etc. While there are commercial solutions from Google, ARM, Kaspersky, Microsoft, Siemens, Intel, etc.

An analysis of works in this area, see Section 1.2, showed that at the moment there is no general approach for the design of secure microcontroller-based systems, and the existing solutions have a limited scope and are not without drawbacks. Therefore, the thesis presents the research aimed at developing the original model-methodological apparatus for the design of microcontroller-based physical security systems. This apparatus will provide a possibility to combine various design techniques on the basis of hierarchical relational model transformation algorithms, while being modular and extensible, taking into account the physical layer of the system, working with abstract system representation and being based on a trade-off between the security of the solution and expended resources.

Scientific task. Development of the model-methodological apparatus for the design of microcontroller-based physical security systems protected from cyber-physical attacks.

Object of the study. Microcontroller-based systems, physical security systems, secure systems design process, model of attacker, attack actions modeling, microcontroller-based devices.

Subject of the study. Models, algorithms and methodologies for the design of microcontroller-based physical security systems protected from cyber-physical attacks.

Goal of the study. Enhancing the protection of microcontroller-based physical security systems from cyber-physical attacks by increasing the number of analyzed parameters during their design process. These parameters are described in more detail in Section 1.5, where the research problem statement is presented.

Objectives:

1. Analysis of the main security issues of microcontroller-based systems. Analysis of the place and role of the design approaches in ensuring the security of such systems. Analysis of the main features of microcontroller-based physical security systems.

2. Development of models of the elements of microcontroller-based physical security systems, including security ones.
3. Development of the hierarchical model of the attacker that allows distinguishing between attackers based on their types of access, knowledge and resources.
4. Development of the model of attack actions that allows checking the possibility of implementation of different classes of attacks based on the attacker's parameters (subject) and system elements (object).
5. Development of the model of microcontroller-based physical security system, which is an extendable set-based hierarchical relational unification of models of system elements, attacker and attack actions.
6. Development of the set of algorithms for the design of the extendable set-based hierarchical relational model of microcontroller-based physical security systems.
7. Development of the methodology for design of microcontroller-based physical security systems, combining the set of algorithms and extendable set-based hierarchical relational model into a single automated approach with minimal operator involvement.
8. Development of the software implementation of the design methodology for microcontroller-based physical security systems, its experimental evaluation.

Scientific novelty. In this work there are multiple scientifically novel results: the extendable set-based hierarchical relational model, the algorithm for the formation of requirements for the system, the algorithm for the formation of the system components composition, the algorithm for the design of the abstract model of the system, the algorithm for the design of the detailed model of the system and the methodology for the design of microcontroller-based physical security systems. Let's consider them in more detail.

Unlike existing solutions, *the extendable set-based hierarchical relational model* represents a microcontroller-based physical security system instead of representing individual microcontroller-based devices. Such functionality neutralizes the disadvantages of analogs in terms of designing devices separately from their interaction with each other. Moreover, this model is modular, extensible and hierarchical, has a strong focus on security of the resulting solution as well as considers security elements as an integral part of the designed system. The extension of the model is possible by introduction of the new levels of abstraction. The modularity of the solution provides the possibility to change its individual parts without the need to change the model completely. For example, the parameters of the attacker's model or available classes of attacks can be updated. The hierarchical nature of the model allows direct (from the whole system to individual elements) and reverse (from an individual element to the system as a whole) transitions.

The novelty of *the algorithm for the formation of requirements for the system* is in retrieving a list of microcontroller-based system devices, communications available to them, as well as requirements for them only on the basis of system tasks, while the list of attack actions that are possible for the attacker is retrieved in accordance with the type of access, knowledge and resources the attacker has.

Unlike other solutions, *the algorithm for the formation of the system component composition* is retrieving abstract elements and sub-elements of the designed microcontroller-based system in accordance with the requirements, device base and already retrieved elements, while security elements are represented as abstract elements, sub-elements, and recommendations for the system implementation.

The novelty of *the algorithm for the design of the abstract model of the system* is in taking into account complex dependencies between the elements of microcontroller-based systems, namely, their hierarchy, nesting, communications, conflicts and requirements. Moreover, this algorithm is not limited to specific platforms and architectures and because of its abstract nature reduces the number of parameters to be searched, thereby increasing the work speed of the solution.

Unlike existing solutions, *the algorithm for the design of the detailed model of the system* makes it possible to form a step-by-step process of detailing the abstract representation of microcontroller-based physical security systems in accordance with the hierarchy and mutual dependencies of their elements. Moreover, this algorithm calculates the parameters of the system devices based on the parameters of their elements as well as the parameters of the system based on the parameters of its devices. This algorithm does not replace the abstract model of the system, but expands and complements it.

The novelty of *the methodology for the design of microcontroller-based physical security systems* lies in a new approach to the design, which allows combining various design techniques on the basis of hierarchical relational model transformation algorithms. Moreover, the suggested approach is modular and extensible, takes into account the security of the physical layer of the system, works with the abstract system representation and is looking for a trade-off between the security of the final solution and expended resources. Also, unlike existing solutions, the methodology has a strong focus on security. It is aimed at ensuring the protection of the system against attacks at the design stage, considers security components as an integral part of the system and checks if the system can be designed in accordance with given requirements and limitations.

Theoretical and practical significance. The obtained theoretical results are very important for such fundamental issues as ensuring information security of microcontroller-based systems, and are aimed at expanding and improving the existing model-methodological apparatus for the design of such systems. The

practical significance of the obtained results lies in the fact that the system based on the proposed models, algorithms and methodology can be used as a tool for designing secure systems based on microcontrollers, thus, avoiding errors in the early stages of their life cycle. Also, this tool can be used by users and system administrators to analyze the security status of systems and devices that are within their area of responsibility. And although at the moment the results obtained are applicable mainly for physical security systems, it is possible to extend the design methodology to other classes of microcontroller-based systems.

Research methodology and methods. Methods and approaches of the system analysis, representation and description of knowledge, analytical, simulation, set-theoretical and ontological modeling, risk analysis, theory of decision-making support, solution of optimization problems.

The main findings:

1. *The extendable set-based hierarchical relational model of microcontroller-based physical security systems protected from cyber-physical attacks* and its elements, namely, models of hardware, software and software-hardware elements, interfaces, protocols and links between system elements at its various levels, models of attacker and attack actions.
2. *The set of algorithms for the design of extendable set-based hierarchical relational models of microcontroller-based physical security systems protected from cyber-physical attacks*, namely, algorithm for the formation of requirements for the system, algorithm for the formation of the system components composition, algorithm for the design of the abstract model of the system and algorithm for the design of the detailed model of the system.
3. *The methodology for the design of microcontroller-based physical security systems protected from cyber-physical attacks*, that combines the set of algorithms and the extendable set-based hierarchical relational model into a single automated approach with minimal operator involvement.
4. *The software implementation of the methodology for the design of microcontroller-based physical security systems protected from cyber-physical attacks*, that validates its correctness based on the design of a system of mobile robots for perimeter monitoring.

Validity and reliability of the study. The validity and reliability of the results obtained is confirmed by their approbation at conferences of Russian and international levels, as well as victories in research competitions. The results obtained are accompanied by logical conclusions based on the results of the experimental evaluation. The models, algorithms and methodology proposed by the author for the design of systems based on microcontrollers are based on modern approaches used in the field of information security, and the methods of their application are correct and justified.

Implementation of research results. The research results presented in this work were used in the following research and development projects:

1. "Models, techniques and methodology for design and verification of secure cyber-physical systems". Research grant #19-37-90082 "PhD students" of Russian Foundation of Basic Research, 2019-2022.
2. "Security Aspects of Cyber-Physical Systems". Research grant #19-17-50205 of Russian Foundation of Basic Research, 2019-2020.
3. "Research and development of an integrated security system based on embedded intelligent microcontrollers". Grant from the Fund for Assistance to the Development of Small Forms of Enterprises in the Scientific and Technical Sphere (Fund for Assistance to Innovation). START-2 project. Contract #2485GS2/22645 dated 04/11/2018, 2018-2019.
4. "Development of methods for vulnerability detection for human-computer interaction interfaces of the Smart City transport infrastructure". Research grant #19-29-06099 of Russian Foundation of Basic Research, 2019-2022.
5. "Methods, Models, Methods, Algorithms, Protocols and Applications for ensuring Information Security of Cyber-Physical Systems". NIR-FUND #717075 of ITMO University, 2017-2019.
6. "Research and development of an integrated security system based on embedded intelligent microcontrollers". Grant from the Fund for Assistance to the Development of Small Forms of Enterprises in the Scientific and Technical Sphere (Fund for Assistance to Innovation). START-1 project. Agreement #1327GS1/22645 dated 06/16/2016, 2016-2017.
7. "Incident management and counteraction against targeted cyber-physical attacks in distributed large scale mission critical systems taking into account cloud services and networks of the Internet of Things". Research grant #15-11-30029 of Russian Science Foundation, 2015-2017.

In research grants #19-37-90082 "PhD students" and #19-17-50205 of Russian Foundation of Basic Research I was lead researcher, while in START-1 and START-2 projects of Fund for Assistance to Innovation I was lead developer of microcontroller-based devices. In other projects (#19-29-06099, #717075 and #15-11-30029) my role was to research and develop an architecture and a prototype of different microcontroller-based systems that are secure by design.

Moreover, the research results presented in this work are used by the department of secure communication systems of the federal state budgetary educational institution of higher education "St. Petersburg State University of Telecommunications named after prof. M.A. Bonch-Bruevich" in the educational process of the direction of training 10.03.01 "Information security" within the discipline "Fundamentals of designing secure infocommunication systems" (work program No. 21.05/446-D) when giving lecture courses, conducting practical exercises and laboratory work.

Approbation of research results. The main results of this research were presented at a number of international and Russian conferences, including:

1. XII Saint-Petersburg Interregional conference Information security of regions of Russia: “Algorithm for the formation of the component composition of a secure microcontroller-based system” [147].
2. 28th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing: “SEPAD – Security Evaluation Platform for Autonomous Driving” [148].
3. XVII St. Petersburg International Conference Regional Informatics: “Requirements for the methodology for design and verification of secure cyber-physical systems” [149].
4. IV Interregional Scientific-Practical Conference Advanced National Information Systems and Technologies: “Approach to the formation of requirements in the design process of secure cyber-physical systems”, “Approach to the formation of specifications for secure cyber-physical systems” [150, 151].
5. IX International Scientific, Technical and Scientific Methodological Conference Actual Problems of Information Telecommunications in Science and Education: “An attacker model for a modern cyber-physical system” [152].
6. XI Saint-Petersburg Interregional conference “Information security of regions of Russia”: “The aspects of the verification of secure cyber-physical systems”, “Application of modeling for verification of cyber-physical systems security” [153, 154].
7. 10th IFIP International Conference on New Technologies, Mobility and Security: “Design and verification methodology for secure and distributed cyber-physical systems” [2].
8. 13th International Symposium on Intelligent Distributed Computing: “The Integrated Model of Secure Cyber-Physical Systems for their Design and Verification” [155].
9. The 1st IEEE International Conference on Industrial Cyber-Physical Systems: “A Technique for Design of Secure Data Transfer Environment: Application for I2C Protocol” [133].
10. The 3rd International Symposium on Mobile Internet Security: “Secure Communication in Cyber-Physical Systems” [156].
11. X St. Petersburg Interregional Conference Information Security of the regions of Russia: “The approach to design of secure systems based on embedded devices” [157].
12. 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: “Design Lifecycle for Secure Cyber-Physical Systems based on Embedded Devices” [158].

Personal contribution. The results presented in this thesis were obtained personally. The author independently analyzed problematic issues of ensuring the information security of microcontroller-based systems, determined the place and role

of approaches to their design, developed models, algorithms and methodology as well as their software implementation to validate the correctness of scientific results using the example of a system of mobile robots for perimeter monitoring.

Publications. The main results obtained during the work on the thesis are published in 22 articles of which 15 publications in journals peer-reviewed by Web of Science or Scopus, 7 publications in journals from the list of Russian Higher Attestation Commission. In addition, 11 certificates of state registration were received, namely 8 computer programs and 3 databases.

The main publications and certificates:

1. Design of secure microcontroller-based systems: application to mobile robots for perimeter monitoring. Sensors. 2021. Accepted 08.12.2021. (Scopus, WoS, Q1) [159]
2. Design and verification of a mobile robot based on the integrated model of cyber-physical systems // Simulation Modelling Practice and Theory, Vol. 105, 2020. DOI: 10.1016/j.simpat.2020.102151. (Scopus, WoS, Q2) [104]
3. Design Technique for Secure Embedded Devices: Application for Creation of Integrated Cyber-Physical Security System. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), Vol.7, No.2, June 2016. P.60-80. DOI:10.22667/JOWUA.2016.06.31.060. (Scopus, WoS, Q2) [60]
4. The application of the methodology for secure cyber-physical systems design to improve the semi-natural model of the railway infrastructure // Microprocessors and Microsystems, November 2020, P. 103482. ISSN 0141-9331. DOI: 10.1016/j.micpro.2020.103482. (Scopus, WoS, Q3) [160]
5. Problematic Issues of Information Security of Cyber-Physical Systems // Informatics and Automation. Vol. 19. No. 5. 2020. P. 1050-1088. DOI: 10.15622/ia.2020.19.5.6. (Scopus, Q3) [161]
6. Application for the design of secure microcontroller-based physical security systems. Federal Service for Intellectual Property. Certificate #2021680236. Registered in the Computer Program Registry 08.12.2021. [162]
7. Database for the design of secure microcontroller-based physical security systems. Federal Service for Intellectual Property. Certificate #2021622496. Registered in the Computer Program Registry 15.11.2021. [163]
8. Component of traffic generation for cyber-physical systems based on I2C protocol. Federal Service for Intellectual Property. Certificate #2018664325. Registered in the Computer Program Registry 14.11.2018. [164]
9. Repository for heterogeneous data from the hardware elements of the smart home. Federal Service for Intellectual Property. Certificate #2017620996. Registered in the Database Registry 01.09.2017. [165]
10. System for support and management of database of the room access control and management system based on the contactless smart cards. Federal

Service for Intellectual Property. Certificate #2016612543. Registered in the Computer Program Registry 01.03.2016. [166]

11. Database of the logging server of a secure access control system for Smart House model. Federal Service for Intellectual Property. Certificate #2016621608. Registered in the Database Registry 29.11.2016. [167]

Structure and volume. The thesis contains an introduction, 6 chapters and conclusion. The main results are presented on 216 pages. The full volume of the thesis is 248 pages with 113 figures, 20 tables and 3 appendices. The list of references contains 167 titles.

Summary. Chapter 1 provides a systematic analysis of the main issues of ensuring the information security of microcontroller-based systems. The place and role of design techniques in ensuring such systems security are indicated. requirements for the methodology for design of microcontroller-based physical security systems are formulated. Research problem statement is performed.

Chapter 2 provides a description of the methods for the evaluation of the design methodology for microcontroller-based physical security systems. The evaluation is done according to the time and resource consumption as well as validity of the obtained results.

Chapter 3 presents the extendable set-based hierarchical relational model of microcontroller-based physical security system as well as models of its elements, namely, hardware, software and software-hardware elements, protocols, interfaces and links between elements at its various levels, attacker and attack action models.

Chapter 4 presents the set of algorithms for design of extendable set-based hierarchical relational models as well as the methodology for design of microcontroller-based physical security systems.

Chapter 5 presents software implementation of the methodology for design of microcontroller-based physical security systems. Its architecture, database structure, main functions of the source code and interface are described.

Chapter 6 presents experimental evaluation of the software implementation of the methodology for design of microcontroller-based physical security systems. The correctness of the obtained results is validated based on the design of a system of mobile robots for perimeter monitoring.

Chapter 1. Systematic analysis of the main issues of ensuring the information security of microcontroller-based systems

This chapter provides a systematic analysis of the main issues of ensuring the information security of microcontroller-based systems and indicates the place and role of design techniques in ensuring such systems security. Main features of microcontroller-based physical security systems are discussed. Requirements for the methodology for design of microcontroller-based physical security systems are formulated as well as the research problem statement is performed.

1.1. Main issues of ensuring the information security

Microcontroller-based systems have become an integral part of our lives: from electricity, manufacturing and transportation, to medicine, commerce and personal use [1]. Thus, ensuring the security of such systems is a critical task, which has not yet been fully solved [2]. This is supported, for example, by the news about growing botnets of smart microwaves and refrigerators used to carry out DDoS attacks, as well as hacking of isolated networks of critical enterprises through smart sensors and cameras [3]. This also determines the high relevance of the chosen topic.

Let's consider the main issues of ensuring information security that are typical for microcontroller-based systems in more detail. To do this, we asked the following research questions:

1. What is the object of the attack?
2. Who is the subject of the attack?
3. What are the intentions of the attackers?
4. What is the way to implement the attack?
5. What methods and means of protection can be applied?

As an answer to the first question in Section 1.1.1, the definition and classification of microcontroller-based systems are proposed. This classification allows one to assess the criticality of the system or its elements in accordance with the business processes that depend on them, the complexity in accordance with the functionality, and connectivity in accordance with the interfaces and data transfer protocols used. In addition, this classification allows one to take into account the social aspect of the system in accordance with the involved personnel and potential users.

A classification of attackers is being developed to answer the second and third questions in Section 1.1.2. This classification makes it possible to assess the capabilities of attackers in accordance with their type of access, level of knowledge and available resources. In addition, the proposed classification makes it possible to take into account possible intentions of attackers, including those related to violation

of confidentiality and integrity of information, as well as violation of the availability of devices and interception of control over them.

As an answer to the fourth question in Section 1.1.3, a classification of attack actions is presented. This classification makes it possible to establish the relationship between the attacker and attack actions in accordance with the knowledge and resources necessary for the attacker to implement them, as well as the purpose to which their use corresponds. In addition, this classification establishes the relationship between attack actions and components of microcontroller-based systems, in accordance with which they can be implemented.

As an answer to the fifth question a classification of methods and means of protection is proposed in Section 1.1.4. This classification makes it possible to assess the possibility of implementing attacking actions.

At the same time, the answers to the above questions, just like the classifications proposed as answers to them, are interconnected, see Figure 1.

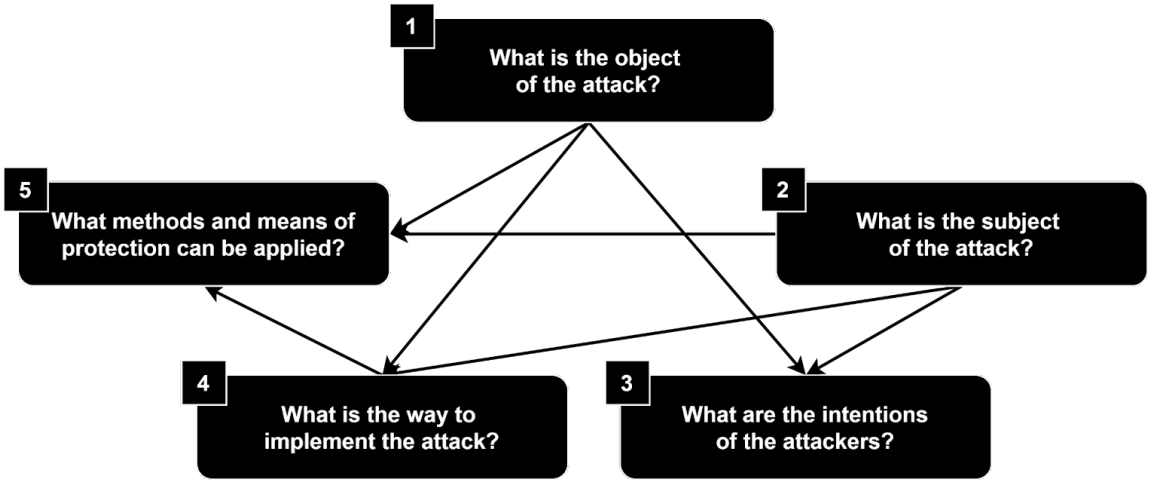

Figure 1. Relationship between problematic issues of information security

Directed arrows show the relationship between the questions. At the same time, the process of building relationships is based on two main concepts of information security: subject and object of the attack. Then, based on information about the target of the attack and the attacker, one can guess the target of the attackers, as well as the tools and approaches they use. In addition, when the information about the object of attack and the attacker is expanded with data about the tools and approaches used by the attacker, it becomes possible to suggest effective countermeasures. Note that the figure shows a direct relationship between questions, while indirect relationships are not displayed — otherwise there would be an all-to-all relationship. Each of these questions, as well as the answers to them, will be discussed in more detail in the following sections.

1.1.1. Definition and classification of microcontroller-based systems

The use of cyber-physical systems is becoming more and more widespread and in demand, since the integration of information technologies and devices for interacting with physical processes and objects is implemented in them. It is important to note that in the scientific literature there is no single definition of such systems and in a number of works there are various descriptions of it. The term cyber-physical system was first proposed in 2006 to denote complexes consisting of natural objects, artificial sub-systems and controllers [4]. In addition, the popularization of this term is associated with the project Industry 4.0 [5], which is based on the introduction of smart systems into the industry. For example, [6] provides an overview of various types of systems and related processes of transition from mechatronics to cloud-based IoT systems. As a rule, the following systems are referred to as cyber-physical [7]: production management systems, Internet of Things, smart home, robotic systems and unmanned vehicles. Microcontroller-based systems are a subset of cyber physical systems.

In [8], a microcontroller-based system is defined as a new type of system that is the result of combining embedded software systems connected, on the one hand, with their physical environment using sensors and actuators, and on the other hand, with global networks such as the Internet with its data and services. According to [9], a microcontroller-based system is a complex technical system that integrates sensor technology and computing, communication and control technologies. The hardware and software of the system are closely linked through the network, forming four processes: data collection, data analysis, decision making and execution. The work [10] uses the concept of cyber-physical space to denote a conditional environment in which physical objects and their informational entities exist in an inseparable connection. And in [7], the concept of a microcontroller-based system is presented as a convenient concept for representing technological systems as a result of the integration of physical processes and the information environment.

Summarizing, the following characteristics can be distinguished, which make it possible to classify the system as microcontroller-based:

1. Integration of information technologies with the physical environment.
2. Existence of processes for collecting, storing, analyzing and providing data.
3. Availability of a reliable data transfer environment between system elements.
4. System contains only microcontroller-based devices.

This means that a microcontroller-based system can be defined as a system that performs the functions of collecting, storing, analyzing and providing data from microcontroller-based devices interacting with physical processes and objects, as well as their close integration with information technology within a reliable data transfer environment.

Information security of a microcontroller-based system means ensuring the integrity, confidentiality and availability of processed data, as well as infrastructure and associated physical processes. Information security of a microcontroller-based system can also be understood as the security of information and information resources of this system from various kinds of threats.

Just like in the definition of microcontroller-based systems, there is not a single classification of it in the scientific literature. In a generalized form, the main attributes of the classification of such systems can be represented as follows: *complexity* in accordance with the functionality and components used; *connectivity* in accordance with the interfaces and data transfer protocols used; *criticality* in accordance with system-dependent business processes; *social aspec*t in accordance with the nature of the interaction of the system with users and operators. Understanding these attributes provides insight into a microcontroller-based system, helping to determine what an attacker is targeting and what capabilities an attacker uses when attacking that system. Let's consider each of the presented attributes in more detail.

Assessment of the complexity of a microcontroller-based system can be carried out in accordance with its functionality and components used. These parameters are most actively studied in works related to their design. In this case, the components of the system are usually divided into different levels, depending on the functionality of the elements of each layer.

For example, the authors of [11] have proposed a service-oriented architecture for microcontroller-based systems, consisting of physical, network and service layers. The study [12] distinguishes the perception level, network level and application level. The task of the physical layer, or the level of perception, is to reliably read information from sensors. The network layer provides ubiquitous data access and transmission. At the service or application level functions for collecting, storing, processing and presenting data are performed.

In [13, 14], the architecture of a microcontroller-based system is proposed, consisting of five levels, which contain: *connection level* — collection of all types of data from sensors and system controllers; *network layer* — analysis of heterogeneous data in order to determine meaningful information; *cybernetic level* — the central information node in the architecture, realizing data analysis and control of the system; *knowledge level* — presentation of knowledge to users, visualization and decision making; *configuration level* — feedback between levels, performance of central dispatching control functions.

Also a common representation of the architecture of microcontroller-based systems is the 7-layer structure of the ISO/OSI model — from the physical to the application layer [15, 16]. Thus, the elements of the system can be classified according to their functionality as well as from the place occupied in the overall architecture.

Microcontroller-based systems can also be classified depending on the processes involved in processing the data they use. For example, in the work [17], was proposed to classify these systems by the semantic level of the data used for operation: *connection level* — data provided by sensors; *conversion level* — data from sensors, after their preliminary processing and aggregation; *cybernetic level* — data from other systems; *knowledge level* — processing of sensor data based on modeling and differential analysis to diagnose the state of the system; *configuration layer* — using incoming data for adaptation and reconfiguration.

In addition, according to [18], the complexity assessment can also be carried out on the basis of the following structural features of microcontroller-based systems: *the number of control loops* — with one control loop and multiple control loops; *the structure of the control loops* — single-level and hierarchical; *quantitative composition of elements* — fixed and variable; *the qualitative composition of the elements* — homogeneous and heterogeneous; *dynamics of behavior* — adaptive and self-organizing. At the same time, adaptation and self-organization means a reaction to external influences, the ability to predict upcoming changes in the external environment, conducting internal testing and improving one's own organization not only under the influence of external factors, but also in the case of conditionally stable work.

Note that in the field of artificial systems there is no clear boundary dividing simple and complex systems. At the same time, there are two main ways to assess the complexity of systems [19]. The first is related to the amount of information required to describe the system and determines its descriptive complexity. Such an assessment is possible on the basis of quantitative parameters of the system, such as the number of elements, connections and hierarchical levels, as well as non-overlapping system functions [20]. The second method makes it possible to estimate the complexity of cognition of the system and is associated with the amount of information required to reduce the system's uncertainty measure. At the same time, the descriptive complexity and the complexity of cognition complement each other — an increase in one complexity entails an increase in another. The role of the classification of microcontroller-based systems is to limit the ways of describing such systems, which provides the basis for their assessment.

The connectivity assessment of a microcontroller-based system can be carried out in accordance with the interfaces and data transfer protocols used in it. This assessment affects one of the most important elements of any system — the process of organizing reliable data exchange between its components. At the same time, existing telecommunication technologies include both data transmission algorithms and tools of their implementation up to physical communication channels.

In [18] to assess the connectivity of microcontroller-based systems, it is proposed to use such features as geographical distribution and openness of the system. With regard to geographical distribution, there are: *centralized systems* — systems located within the boundaries of one physical object; and *distributed systems* — systems located on several interconnected objects. The openness of the system determines the nature of the use of internal and external (global) networks and classifies the microcontroller-based system as a *closed-type* system if only the internal communication environment is used for its operation, and an *open-type* system if the system requires access to the global Internet.

In [17] to assess the connectivity of microcontroller-based systems, it is proposed to use the technologies and communication standards. Technologies characterize the devices used by the system to interact with physical objects or processes, while standards characterize the process of interaction of system elements with each other, indicating the protocols and interfaces. Protocols are divided into high-level, low-level and inter-level, and for the classification of interfaces it is proposed to use various features that characterize the communication topology, the format and mode of data transmission, as well as the functional purpose of the network.

The protocols and interfaces used can be conditionally divided into wired and wireless. Wireless sensors and actuators play a central role in the design of modern microcontroller-based systems. In such complex heterogeneous systems, communication links must meet requirements of bandwidth, latency, and range as well as low power consumption. In [21] the most recent wireless standards are covered, such as: NFC, UHF RFID, ZigBee, Z-Wave, EnOcean, Bluetooth, Wi-Fi, 3GPP, NB-IOT, LoRa and SigFox. At the same time, the following network topologies are distinguished: star, tree, mesh and honeycomb.

The most common wired interfaces for data transfer between microcontroller-based devices include UART, SPI, I2C, Ethernet, 1-Wire, Modbus, and CAN [22, 23]. Each of the listed interfaces has a number of features that affect the data transfer rate, power consumption, and available additional functions: for example, the functions of addressing and identifying connected devices. At the same time, for these interfaces, their hardware implementations are widespread, which led to their integration into most modern devices based on microcontrollers.

Note that the global informatization of various spheres of human life contributes to both the development of existing specifications for network exchange protocols and the emergence of new protocols. At the same time, for devices of microcontroller-based systems there is a tendency to use proprietary protocols — protocols with unregulated (at least publicly available) specifications. This situation is mainly related to the desire to protect the intellectual and commercial property of companies as well as to complicate the conditions for the analysis of network protocols by third-party researchers. This means that often traffic in

microcontroller-based systems can be characterized as traffic of large volume, high heterogeneity and undefined structure [24].

The criticality assessment of a microcontroller-based system can be carried out in accordance with the business processes that depend on it. To carry out this assessment, business process models are often used, as well as an analysis of potential threats and vulnerabilities for subsequent risk assessment and the selection of countermeasures. In this case, risk is defined as the ability of a particular threat to use the vulnerability of one or more assets to harm the organization [25]. In turn, assets can represent tangible assets, information, software and hardware, personnel and intangible resources of value to the organization.

By definition, a critical information infrastructure is a set of automated control systems for production and technological processes of critical objects, as well as information and telecommunication networks that ensure their interaction [26]. Thus, these objects can include microcontroller-based systems operating in the spheres of health care, science, transport, communications, energy, finance, defense and industry. An analysis of the field of application of microcontroller-based systems is presented in [27-29]. Let's consider them in more detail.

In [27] the following areas of application of microcontroller-based systems are distinguished: public security, retail trade, transport, industry, healthcare, smart home, construction and energy. For each area, an end user is identified and examples of devices are provided. The authors of [28] review existing solutions in the design of microcontroller-based systems, which allows highlighting the following areas of application: automotive systems and transport, medical systems, smart homes and buildings, social networks and gaming systems, planning systems, control systems, power systems, systems surveillance, industrial systems, aerospace systems, search systems, ecological systems, construction systems, robotic systems, and water distribution systems. The article [29] examines the main components of the modern intellectual environment, namely such concepts as smart home, smart health, smart city and smart factory. At the same time, these concepts are compared with current communication solutions in the field of microcontroller-based systems. This paper also provides an overview of communication technologies and architectures of such systems, and in the conclusion discusses the problems that remain open for research.

Industrial microcontroller-based systems are characterized as a combination of autonomous and coordinated elements (from machines to logistics networks), connected to each other in accordance with a set of goals at all levels of production and capable of making decisions in real time [30]. At the same time, the benefits of implementing such systems are being explored everywhere. For example, [31] shows the process of introducing the principles of microcontroller-based systems into the industrial sector by organizing the work of enterprises in the framework of

technologies such as smart manufacturing and digital factories. The works [32, 33] show the advantages of interaction between humans and robotic systems in a hazardous environment. The article [34] describes transport microcontroller-based systems, their basic principles of organization and functioning. In [35], a paradigm of a microcontroller-based building system is proposed, which is a finite set of functional components such as building objects and complexes, as well as computing resources integrated into the included physical processes. A number of works [36, 37] provide studies of medical microcontroller-based systems to improve the efficiency and safety of healthcare.

Note that the criticality of a microcontroller-based system is characterized by the consequences of complete or partial failure of both the entire system and its individual elements. These consequences include both financial and reputational damage and a threat to human life and health. One of the ways to represent criticality is a vector of the following components: reliability, failure consequences, the ability to reduce the likelihood and severity of consequences [38]. The ranking of the elements of the system according to the degree of criticality depends on the system type, selected particular indicators as well as available expert information.

The criticality of information processed in microcontroller-based systems as a rule is determined by the owner of the system and may depend on various parameters. For example, the criticality of information can be affected by its need for the correct functioning of the system as well as damage from its loss, modification or leakage. Criticality can be calculated using both qualitative and quantitative indicators [39].

In [40] a classification of information assets in accordance with the requirements for confidentiality, integrity and availability is proposed. With regard to confidentiality, the authors highlight information that is restricted for distribution according to the requirements of the law and organization as well as open information. With regard to integrity, information is distinguished based on the damage the violation of its integrity can lead — significant, moderate or insignificant damage — as well as information, the integrity of which is not required. With regard to accessibility, authors highlight information that is available at any time as well as information that is available with a delay of up to several hours/days/weeks.

Based on the classification proposed in [40], information can be divided into *critical information* — confidentiality must be ensured in accordance with the requirements of the law, violation of integrity can lead to significant damage, information is available at any time; *important* — confidentiality must be ensured in accordance with the requirements of the organization, violation of integrity can lead to moderate damage, information is available with a delay of up to several hours; and *usual* — confidentiality and integrity are not required.

The assessment of the social aspect of a microcontroller-based system can be carried out in accordance with the nature of the interaction of the system with users and operators. At the same time, this area of research gave rise to such a term as the socio-cyber-physical system. It is important to note that the efficiency of the functioning of a microcontroller-based system depends not only on hardware and software, but also on the personnel and users interacting with it. This means that the interests of various social groups should be taken into account both at the level of the formation of the external appearance of the system and in the development of its technical specifications.

Thus, in [41] this fact made it possible to introduce a sign of socialization of the elements of a microcontroller-based system, which characterizes the following types of interaction of the system with society: design, production, purchase/sale, storage, work (operator), maintenance and disposal. And in [17] the human factor feature was introduced, which describes the following types of interaction of microcontroller-based systems with the operator: *autonomy* — the system makes all the necessary decisions without any operator intervention; *automation* — the system guides the operator during tasks, making most of the decisions; *tool* — the operator manages the system and is responsible for most decisions; *management* — the system only provides data to the operator, who makes all decisions.

Microcontroller-based systems often simulate the intellectual capabilities of a person in the tasks of searching, analyzing and synthesizing information about the world around them in order to obtain new knowledge and solve the assigned tasks. Thus, in [18] for such systems, the concept of intellectualization is introduced, which describes the system's ability to learn, gain experience and make decisions. In addition, this work introduces the concept of the dynamics of response to the external world, which is divided into the dynamics of high, medium and low levels. It is assumed that this feature can be used to assess the ability of microcontroller-based systems to work with uncertain and dynamic data, as well as to extract knowledge from accumulated experience. Also, in this work, the concept of a model of perception of the external world is introduced, which describes how objects of a microcontroller-based system perceive the surrounding world: without a model of the external world, with a given model of the external world, or with a model of the external world that is generated during the operation of the system.

Based on the analysis and systematization of the current state of research, as the main attributes of the classification complexity, connectivity, criticality and the social aspect of microcontroller-based systems were chosen. Using these attributes, a classification was built as shown in Figure 2.

This classification allows one to assess the criticality of the system or its elements in accordance with the business processes that depend on them, the complexity in accordance with the functionality and connectivity in accordance with the interfaces

and data transfer protocols used. In addition, this classification allows one to take into account the social aspect of the system in accordance with the involved personnel and potential users. The sufficiency of the classification is confirmed by the analysis of existing scientific and practical works, in which the above attributes are used to determine the type of system.

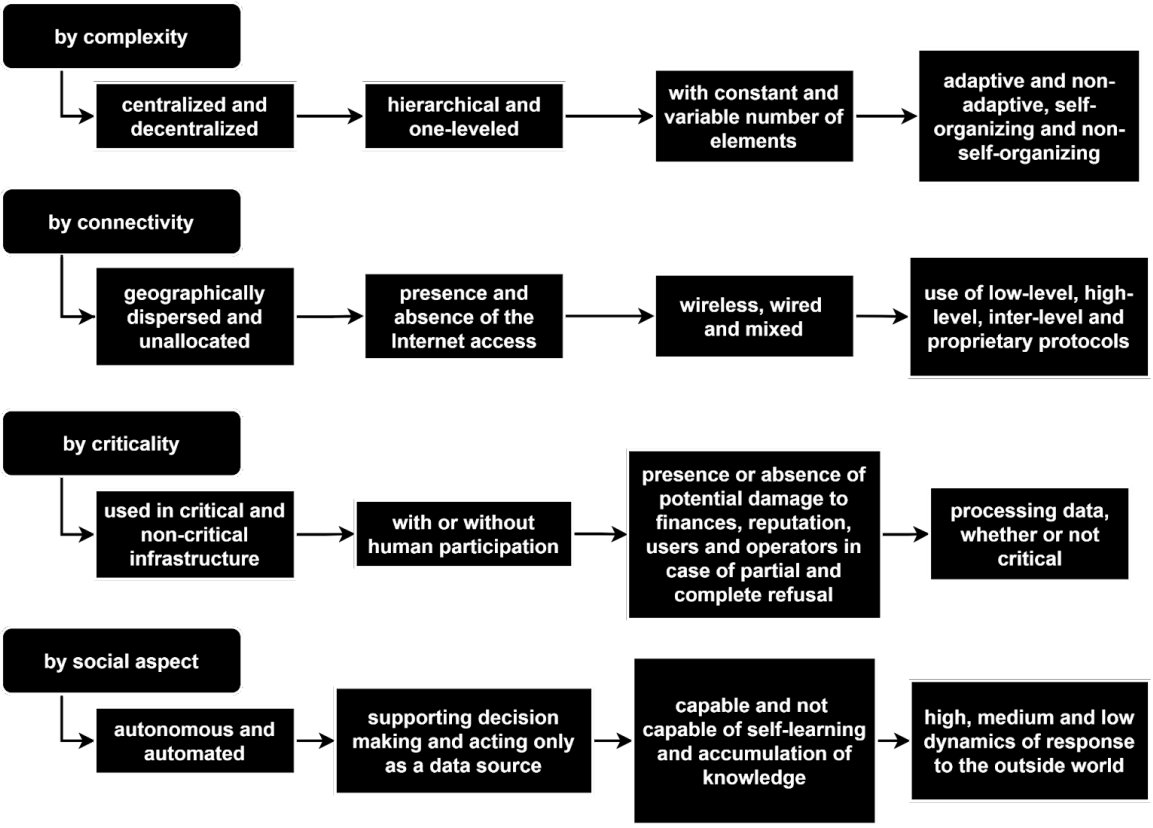

Figure 2. Classification of microcontroller-based systems

For example, in terms of complexity a decentralized single-level self-organizing system with a variable number of elements can be distinguished. In terms of connectivity — a geographically distributed system with access to the Internet, built on the basis of wireless and wired technologies using low-level and high-level protocols. In terms of criticality — a system used in a critical infrastructure with human participation that processes critical information, the failure of which could result in financial damage. With regard to the social aspect, it is an autonomous system that acts as a source of data, is not capable of self-learning and accumulation of knowledge, and has a low dynamic of response to the outside world. Each of the obtained classifications allows us to restrict the way of describing the systems under study and provides a basis for assessing their complexity, connectivity, criticality and social aspect.

1.1.2. Analysis and classification of attackers

An important step in the process of identifying threats to the security of a microcontroller-based system is the identification of persons whose actions can lead to a violation of the confidentiality, integrity or availability of the system and the occurrence of damage. According to the definition in GOST R 53114-2008 [42] an individual or logical object is considered to be an intruder if they accidentally or deliberately committed an action that entailed negative consequences. The attacker's model, or profile, characterizes the possible ways of interaction between the attacker and the target system, in particular, it defines the restrictions for the attacker. The result of the analysis of the attacker's model is an assumption about the types and potential of intruders who can implement security threats for a microcontroller-based system with given characteristics and functioning features.

It is assumed that the classification of attackers will make it possible to assess their capabilities in accordance with the type of access to the system, the level of knowledge, possible intentions and available resources. *The type of access* allows one to distinguish between an external and internal intruder, an ordinary user and an administrator. *The level of knowledge* is a characteristic of the attacker, which indicates his technical skills for initiating and carrying out an attack. Also, this characteristic describes the intruder's awareness of the architecture of the target system and the existing protection measures. *The intent* of the attacker indicates the purpose of the attack on the system. This parameter is difficult to quantify and is very dynamic. An attacker's *available resources* include hardware and software resources that can be used to deploy a specific type of attack.

The main regulatory documents defining the model of an attacker in Russian Federation are “Basic model of threats to the security of personal data during their processing in personal data information systems” [43], “Methodology for determining threats to information security in information systems” [44] and “Methodological recommendations on the development of regulatory legal acts that determine threats to the security of personal data, relevant when processing data in personal data information systems operated in the implementation of relevant activities” [45]. Moreover, internal intruders are divided into eight categories depending on the method of access and access authority: *category 1* — persons who have authorized access to the system and ensure its normal functioning; *category 2* — registered users of the system who have limited access to its resources from the workplace; *category 3* — registered users of the system who provide remote access to its resources; *category 4* — registered users of the system with the privileges of a security administrator for a separate segment of the system; *category 5* — registered users with the authority of the system administrator; *category 6* — registered users with system security administrator privileges; *category 7* — developers of the system software and persons providing its support; *category 8* — developers and persons providing delivery, maintenance and repair of system equipment.

The regulatory document [44] introduces the concept of an intruder's potential, which can be low, medium and high:

- *low potential* — the intruder has information about the vulnerabilities of individual elements of the microcontroller-based system, published in public sources, while using publicly available tools or tools created on his own;
- *medium potential* — the intruder has all the capabilities of low potential intruders and also has an awareness of the protective measures used in the system; in addition, the intruder has information about the vulnerabilities of individual elements of the system and uses freely available software tools to carry out attacks, and also has access to information about the characteristics and features of the functioning of the system;
- *high potential* — an intruder has all the capabilities of an intruder with a medium potential, and can also get unauthorized access to the microcontroller-based system from dedicated communication networks; in addition, an attacker of this type has access to the software and hardware of the system, is well aware of the protection measures used in it, and also has information about system vulnerabilities, conducts research on the attacked system and uses highly specialized tools.

The normative document [45] provides generalized capabilities of intruders, with the main attention being paid to the capabilities of an attacker to attack system protection and its environment: the ability to attack a microcontroller-based system only outside the controlled area; the ability to attack a microcontroller-based system within a controlled area, but without physical access to it; the ability to attack a microcontroller-based system within a controlled area with physical access to it; the ability to attract specialists with experience in the development and analysis of security measures typical for microcontroller-based systems.

It is important to note that in addition to the main regulatory documents, various classifications of attackers are given in a number of studies in the field of information security threat analysis. Let’s consider these works in more detail.

For example, in [46] an overview of research into attacks on microcontroller-based systems, as well as attacker profiling is provided. This review concludes that existing studies can be grouped into two main categories: (1) studies that are using different models of attackers with different properties (for example, one model to describe the insider, the other to describe the state intelligence service); (2) studies that are defining a number of parameters such as the type of knowledge, level or potential of the intruders to distinguish between them within the framework of a single model.

In addition, this paper proposes a generalized classification of attackers, including the following types:

- *amateur* — uses publicly available tools to attack the system and has standard access to hardware, software and Internet connection;

- *internal intruder* — has system privileges (for example, user, supervisor, administrator);
- *hacktivist* — uses his abilities to manifest political activity;
- *cyber terrorist* — a politically motivated attacker who uses his abilities to commit crimes;
- *cybercriminal* — an attacker with extensive security knowledge and skills, whose goals can range from blackmail to espionage and sabotage;
- *faction* — a group of people, sometimes funded by the government, that often aims to attack critical infrastructure systems.

The authors also note that the boundaries between the types of attackers in the above classification are rather blurred, and therefore it can be difficult to identify a real attacker as one specific type. With regard to the goals of the attackers, the authors distinguish: personal, economic, forensic, terrorist and political.

In [47] a classification of attackers on a microcontroller-based system using the example of a water supply management system is provided. In this case, the attacker is classified according to the type of access to the system and capabilities. The authors distinguish the following types of access to the system:

- *type 0* — the attacker does not have direct access to the infrastructure and services of the system, only social engineering methods are available for use;
- *type 1* — the attacker interacts with the infrastructure and services of the system indirectly, providing indirect access to them;
- *type 2* — an attacker affects the system infrastructure or its services directly, while being at a certain distance from the controlled perimeter;
- *type 3* — an attacker has physical access to the system infrastructure, but is not able to investigate and modify internal electronic components;
- *type 4* — the intruder has full access to the system infrastructure and all internal elements and interfaces.

At the same time, the authors distinguish the following levels of attackers' capabilities:

- *level 1* — use of publicly available tools and exploitation of known system vulnerabilities;
- *level 2* — the ability to identify and exploit previously unknown vulnerabilities and develop new tools to influence the target system;
- *level 3* — level 2 capabilities and nearly unlimited resources to carry out attacks.

Thus, the classification proposed by the authors allows us to consider attackers from the point of view of the type of access, resources and knowledge necessary for the successful implementation of attacking actions.

Based on the systematization of the current state of research on such attributes of the classification of attackers as the type and method of access, intentions, knowledge and resources, the classification was built, presented in Figure 3.

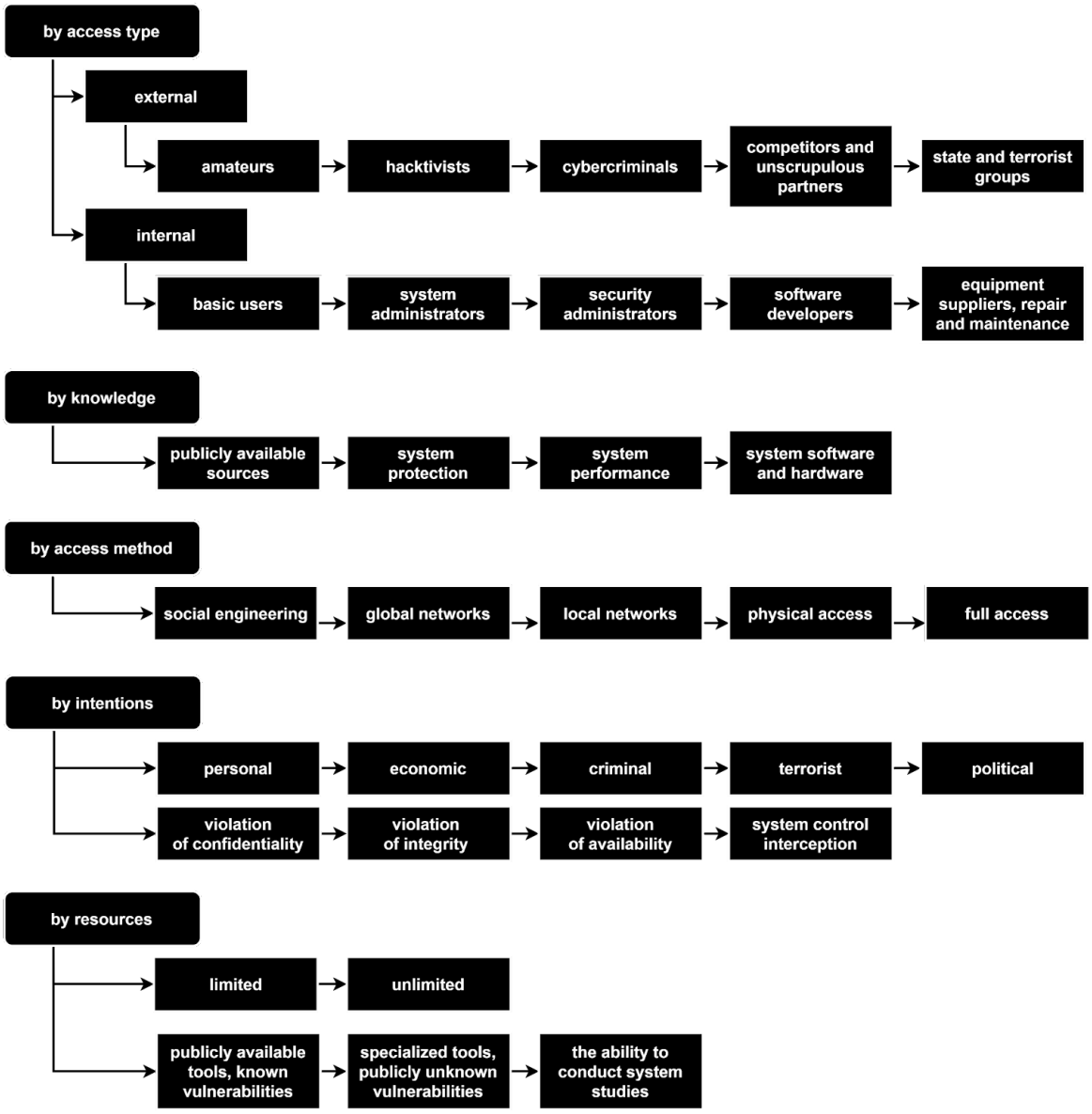

Figure 3. Classification of the attacker

This classification makes it possible to assess the capabilities of attackers in accordance with the type and method of access to the system, the level of knowledge and available resources. In addition, this classification makes it possible to take into account the intentions of attackers, including those related to violation of confidentiality and integrity of information, as well as violation of the availability of devices and interception of control over them.

1.1.3. Analysis and classification of attack actions

An equally important step in the process of identifying threats to the security of a microcontroller-based system is the analysis of actions that can lead to a violation of the confidentiality, integrity or availability of the system. According to the definition in GOST R. ISO/IEC 27000–2012 [48], an attack is an attempt to destroy, disclose, alter, block, steal, gain unauthorized access to an asset or use it unauthorized. At the same time, attacks can occur at different levels of the system, include many stages, be stretched out in time and affect its various elements. And although the variety of attacking actions is actively studied in the scientific community, at the moment there is no single classification of them.

Let's consider the existing work in more detail.

In [49] network attack actions are classified based on resources, topology and traffic:

- *by influence on resources* — directed (denial of service, routing table overflow) and undirected (privilege escalation);
- *by influence on the topology* — reducing performance (substitution of the routing table, "funnel", "wormhole") and isolating ("black hole");
- *by influence on traffic* — eavesdropping (sniffing and traffic analysis) and intercepting (downgrading, spoofing).

In [50], when classifying attacking actions on SCADA systems, the following types of attack actions are distinguished:

- weakening the network perimeter using backdoors;
- exploiting vulnerabilities in the protocols used;
- intercepting control of individual system devices;
- disrupting the database;
- intercepting and modifying network messages;
- modifying the system time for stopping the work of protective equipment.

The study also proposes to divide attack actions into attacks aimed at:

- modifying, intercepting or introducing input data from system sensors;
- changing the system operation process by modifying, intercepting or introducing data at the level of interaction between system controllers;
- modifying system logs;
- intercepting control of individual devices or stopping their work.

In [51] authors propose to represent attack actions as the following data tuples: subject, object, intent, vector and consequences. In this case, the subject of an attack can be an intruder, natural disaster, human factor, errors of the system and supporting infrastructure. The object of an attack can be any element of the system, data transfer environment between them as well as the system as a whole. Intentions can be criminal, intelligence, terrorist or political. Attack vectors are

divided into interception, modification and falsification of data as well as termination of its transmission. The consequences of an attack include compromising the confidentiality, integrity, availability, privacy and reliability of the system.

In [52] the classification of attack actions on microcontroller-based systems is presented. Authors highlight attacks on sensors, computing processes, feedback, data transfer environment and actuators. The examples for each of the listed types of attack actions are:

- *attacks on sensors* — disabling equipment, interrupting power supply, using physical processes for incorrect operation of sensors;
- *attacks on computing processes* — deletion, modification, substitution or forgery of data, worms, viruses, trojans;
- *feedback attacks* — data integrity violation, control interception;
- *attacks on data transfer environment* — deletion, modification, substitution or forgery of data, data loss, sniffing;
- *attacks on actuators* — deletion, modification, substitution or forgery of data, interruption of power supply, modification of hardware and software.

In [53], when analyzing the security of microcontroller-based systems, it is proposed to distinguish attack actions in accordance with the level of the system at which the attack occurs, the element of the system to which the attack is directed, and the intentions of the attacker. At the same time, for each level of the system, the authors presented the main security problems and possible countermeasures.

The authors of [54] also proposed to classify attacks against microcontroller-based systems according to the level of the system: physical, network or application. At the same time, for each level, the authors distinguish the corresponding attack actions:

- *physical level* — disabling equipment, stopping equipment operation, stopping power supply, intercepting electromagnetic signals, denial of service, intercepting and modifying data, stopping data transfer, unauthorized access;
- *network layer* — distributed denial of service, tampering with the routing process, redirection or loss of data, buffer overflow;
- *application layer* — unauthorized access, data leakage, malicious code injection, control interception, virus, trojan and database injection.

In [55] the authors propose to divide attack actions on microcontroller-based systems in accordance with the area of their impact: from interaction with physical devices to various aspects of network interaction (segmentation, topology, technologies used and structure). At the same time, the authors provide the following generalized classification of them: interception and analysis of traffic; leakage of personal data; disabling equipment; remote execution of malicious code; violation of the integrity of the source code of applications; exploitation of vulnerabilities in network protocols; denial of service.

In [56] it is proposed to classify attack actions on microcontroller-based systems according to their object, impact and performed action. For each action, a method and preconditions are distinguished, and for the object and impact, the affected element and the influence on it.

In [57] it is proposed to classify attack actions on microcontroller-based systems in accordance with the object of the attack, the impact on the system and the impact on the person. Let's consider the proposed classification in more detail:

- *object of attack* — data collection, data transfer environment, control system;
- *impact on the system* — physical (incorrect operation, denial of service, slow data processing) and cybernetic (confidentiality, integrity, availability, non-appealability);
- *impact on a person* — emotional impact, influence on acquired experience, physical harm.

In [58] attack actions are divided on the basis of the impact method and the security aspect. At the same time, according to the impact method, there are:

- *informational* — unauthorized access, copying and theft of information, violation of information processing technology;
- *software* — exploiting bugs and vulnerabilities in software, spreading malware, setting bookmarks;
- *physical* — destruction of system devices, theft of media, theft of keys and cryptographic data protection;
- *radio-electronic* — the introduction of devices for intercepting information, intercepting, decrypting, substituting and destroying data in communication channels;
- *organizational and legal* — violation of the law, purchase of outdated programs and devices.

In terms of security aspect, attack actions are divided into ones that are violating confidentiality, integrity and availability.

Based on the analysis and systematization of the current state of research on such attributes of the classification of attack actions as subject and object, impact method, prerequisites and consequences, a classification was built, presented in Figure 4.

This classification makes it possible to establish the relationship between the attacker and attack actions in accordance with the knowledge and resources necessary for the attacker to implement them, as well as the purpose to which their use corresponds. In addition, this classification establishes the relationship between attack actions and elements of microcontroller-based systems in accordance with which they can be implemented.

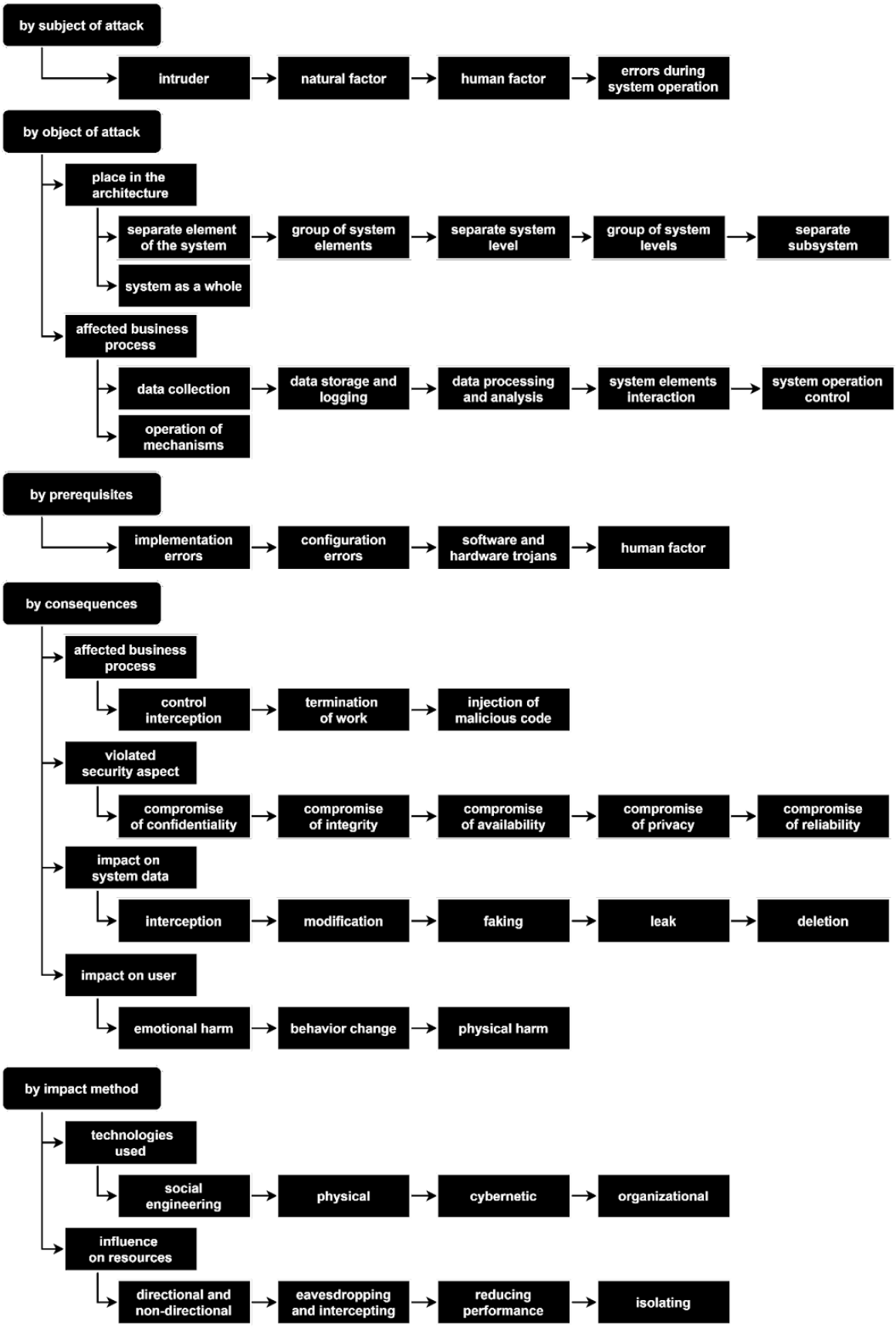

Figure 4. Classification of attack actions

1.1.4. Analysis and classification of methods and means of protection

Since one of the key features of microcontroller-based systems is the close integration of physical processes and information technologies, the number of problems that must be considered when developing security mechanisms for such systems is much higher in comparison with other types of systems. In addition, such systems often have a dynamic infrastructure, heterogeneous data sources and stores, which also increases the complexity of the required protection. At the same time, most of the research in this area is aimed at solving various security problems at each individual level of the architecture of the system, and not for the system as a whole. Let's consider the existing work in this direction in more detail.

In [59, 12] the authors propose to determine the necessary methods and means of protection based on the component composition of the microcontroller-based system. At the same time, these works provide a classification of protection methods in accordance with the level of the system, the protection of which they provide. The authors distinguish the following levels: *level of data collection* — certification, access control, authentication, lightweight data encryption, physical security of devices, environmental monitoring, trust management; *data transfer layer* — reliable routing and data encryption, authentication and key agreement, network access control, attack detection mechanism; *data analysis and processing level* — end-to-end encryption, intrusion detection, trust management, authentication and authorization, data mining, forensics, personal data protection.

Note that protection methods from [12] are referred by the authors to the information field of the system, in addition to which they also distinguish the control field and risk assessment. It is noted that these security mechanisms should be developed taking into account the security of the system as a whole, and not just its individual level. At the same time, this process includes the development of an integrated cross-layer security solution that is capable of working with different methods and means of protection, and also reliably integrates data from different sources.

In [60, 61] the architecture of a microcontroller-based system is presented. This system integrates both physical and information security solutions and consists of the following main parts: *data sources* — include various physical and cyber security systems; *data collection module* — uses various hardware and software interfaces to connect to data sources, while the received data is subject to preprocessing and normalization processes; *data analysis module* — includes various stages of the security event correlation process; *data presentation module* — includes such processes as security assessment, development of countermeasures and generation of reports. Note that, in accordance with the architecture proposed by the authors, the methods and means of protection of the system can be classified in accordance with the problem being solved.

In [7, 62] it is proposed to consider methods and means of ensuring the security of microcontroller-based systems from the point of view of control theory. At the same time, the authors highlight the following features that must be taken into account when designing system protection: the presence of feedback, the presence of an adaptive control loop and the ability to predict the state of the system. On the basis of these features, the authors propose the following classification of methods and means of protection: *static* — the control function does not change over time, the output state of the protected object depends on the constant values of control actions; *active* — the results of experimental testing of the protected object are used to configure the parameters of security systems; *adaptive* — the parameters of security systems are periodically changed to maximize the effectiveness of protection based on the characteristics of the object during the monitoring process; *dynamic* — there is dynamic compensation for unwanted changes in the state of the system during operation. Note that the approach proposed by the authors makes it possible to formulate the task of ensuring the security of microcontroller-based systems as a task of automatic control in conditions of targeted cyber threats in order to ensure the sustainability of functioning.

The authors of [61] propose to analyze the network interfaces and protocols used in a microcontroller-based system to determine the necessary means and methods for protection of the data transfer environment. At the same time, special attention is paid to the process of interaction between the controllers of the system, where in the above experiment, the security of the data bus is ensured by mutual authentication of devices, encryption of transmitted data and reliability — due to dynamic addressing and monitoring of the state of connected devices, the absence of uncontrolled loss of sensor events and integrity checks of transmitted data.

Cisco's security framework [63] has four main components: authentication and identification, access control, network policy and security analytics. At the same time, the basic application of network policy is primarily concerned with ensuring that the traffic entering the network meets the specified rules, including the allowed range of IP addresses and types of traffic. Traffic packets that do not meet the specified rules are recognized as anomalous and should be dropped as close to the network edge as possible, thereby minimizing the risk of impact. As a rule, various methods are used to detect anomalies, the generalized classification of which can be represented as follows: behavioral, statistical and data mining methods [64].

In [65] existing vulnerability assessment methods are examined as well as their role in the security risk assessment process and how they are applied. There are three main groups of methods: quantitative, qualitative and qualitative-quantitative. Quantitative risk assessment methods allow assessing the risk in monetary units and take into account the frequency of undesirable events. Qualitative methods rank risks relative to each other based on asset values, vulnerabilities, threats and

defenses. At the same time, in practice, a qualitative-quantitative approach is mainly used, within which certain ranges of quantitative values are compared.

In [66] the authors review research on the assessment of vulnerabilities in microcontroller-based systems in academic and commercial fields. At the same time, the authors noted that the latter is characterized by a variety of approaches to identifying vulnerabilities, while this is not observed in the academic environment.

In [67] methods for assessing the risks of microcontroller-based systems are considered in terms of the economic effect, which manifests itself even when the attacker's motivation is not financial. The analysis of various models and methods of risk assessment, as well as vulnerability assessment systems is given.

In [68] existing approaches to risk assessment and management are examined in terms of safety, security and their integration. Methods for assessing security risks in microcontroller-based systems include: *fault tree analysis* — a view that allows you to link various legitimate events and errors, the occurrence of which can lead to an undesirable event; *analysis of failures and their consequences* — a structured method of analyzing the safety of a system, which makes it possible to recognize situations that lead to the failure of a system or its individual elements, as well as their consequences; *analysis of criticality and reliability* — a method of analyzing the safety of a system, which makes it possible to assess the degree of criticality and reliability of system processes by studying the consequences of possible deviations; *development in accordance with the model* — a method of developing simulation models of real-time systems and analyzing these models to verify compliance with safety requirements; *analysis of success trees and goals* — a method of analyzing system security based on structural analysis of the reliability and risk of the system; *analysis of emergency processes* — a method of safety analysis based on a set-theoretic model and analysis of situations that leads to an accident.

The work [69] is devoted to the study of the main approaches in the field of risk assessment for potentially dangerous objects. Assessment methods include quantitative assessment using mathematical statistics, expert risk assessment, simulation and their combinations. The study specifies that the assessment of the violation of physical security is carried out for each specific object using the following methods: mathematical modeling of the probability distribution of a risk event; expert assessment by Delphi and ranking methods; numerical integration of the risk function in time and space. This means that the assessment of the security of a microcontroller-based system can be represented as a process of analyzing accumulated data, expert opinion or the work of a mathematical apparatus.

The social aspect of microcontroller-based systems and, accordingly, possible attacks of social engineering lead to the search for methods and means of protection against them. For example, in [70] the phenomena of aggression in the

socio-cyber-physical environment and their impact on the individual and group consciousness of users is studied. The results obtained are proposed to be used in the development of a unified socio-cyber-physical system for managing these processes. The authors note that, in a social network, combining a source with the means and forms of communication used makes it possible to take into account the social effect of the message, which can be used to predict manifestations of aggression, pressure and other destructive phenomena.

In [71] the authors proposed a classification of social-engineering attacks and a possible approach to assessing the security index of corporate networks from the point of view of human behavior. The following basic measures of protection against attacks of social engineering are proposed: availability of information security policy; briefing; monitoring compliance with information security; identity management policy; introduction of biometric access systems.

Based on the analysis and systematization of the current state of research on such attributes of the classification of methods and means of protection as the principle of operation, the object of protection and the problem to be solved, the classification presented in Figure 5 was built.

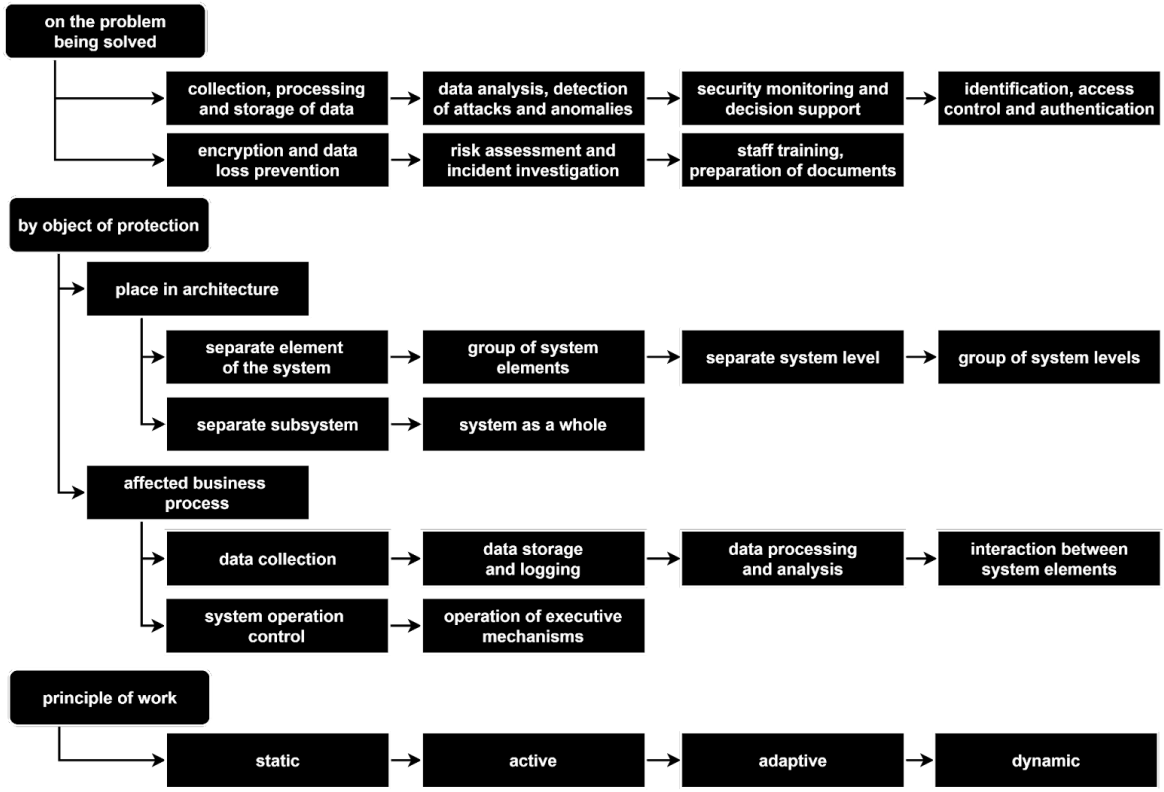

Figure 5. Classification of methods and means of protection

1.2. Place and role of the design techniques

As was shown in Section 1.1, there are many approaches to ensure the information security of microcontroller-based systems. As a rule, they are associated with individual stages of such systems development lifecycle: analysis, planning, design, development, testing, deployment, maintenance and evaluation, see Figure 6.

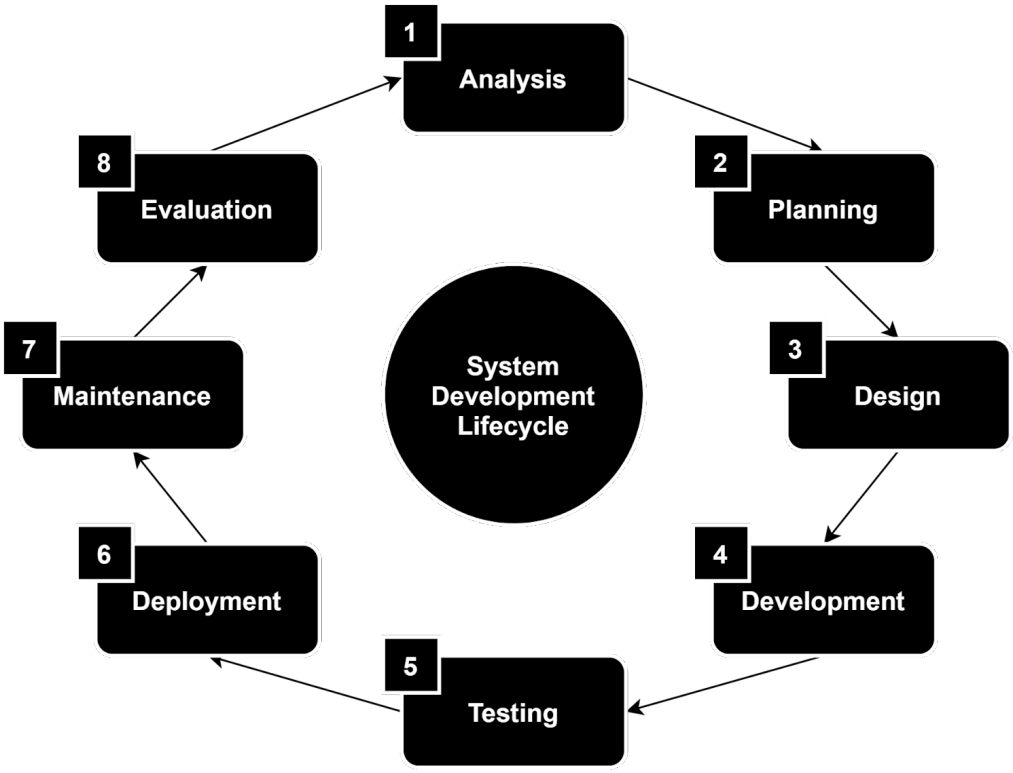

Figure 6. Microcontroller-based systems development lifecycle

The approaches that are discussed in this section are used at the design stage of the microcontroller-based systems development lifecycle and associated with Security by Design. Security by Design is an approach to software and hardware development that aims to reduce the number of possible vulnerabilities and enhance the system's protection against possible attacks. The main idea of the approach is in taking into account security features as a design criterion of products.

The classifications provided in Section 1.1 are making it possible to assess the possibility of implementing attack actions in accordance with the methods and means of protection that are used in the microcontroller-based system. This is possible due to the fact that the classification of methods and means of protection by the object of protection coincides with the classification of attack actions by a similar attribute. Consequently, upon further analysis of the knowledge, resources and capabilities of the attacker, it is possible to conclude about the feasibility of certain attack actions, see Figure 7.

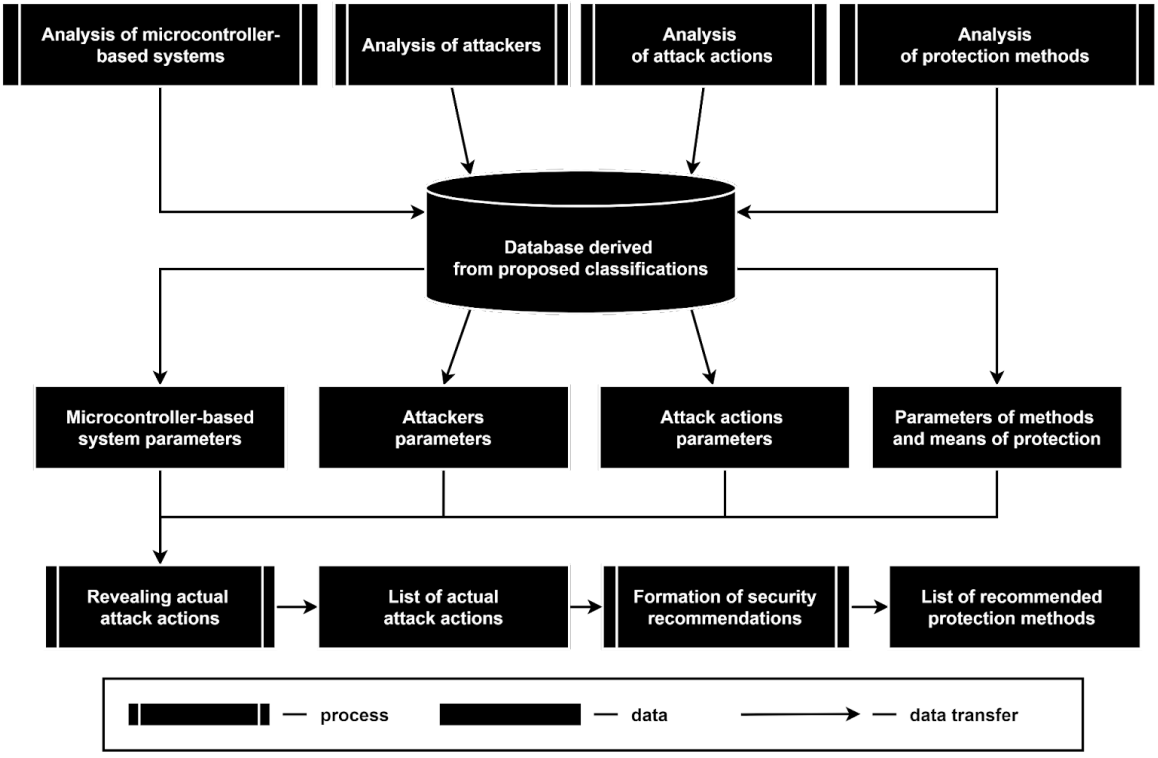

Figure 7. The process of identifying relevant attack actions and recommending methods and means of protection

This means that based on the information about the component composition of a microcontroller-based system, it is possible to determine a list of attack actions to which this system is potentially susceptible. Then, based on the idea of the level of knowledge of the attacker and the resources available to him, this list of attacks can be limited in the same way as if there is information about the methods and means of protection used. All attack actions remaining after these transformations represent a real threat and must be taken into account.

Information about attack actions to which the designed system is potentially susceptible is often used by design techniques to find a trade-off between the level of protection of the resulting solution and resources expended on it.

The task of designing microcontroller-based systems to be secure against attacks is complex, that is why various design techniques have been developed and embedded into practice. Some of them are focused on software, some on hardware, and some on highly specialized areas of the application (automobiles, railway transport, robotics). Let's consider them in more detail.

In [72] it is mentioned that the main goal of design and verification of microcontroller-based systems is to develop a resilient system. According to the authors, the system is resilient when it is designed with 3S features: stability, security

and systematicness. This is achievable through solving five main challenges: dependability, consistency, reliability, cyber-physical mismatch and cyber-physical coupling security. Authors also mentioned that such systems are about tight coupling of cyber and physical objects, so their model contains the following parts: model of the physical process, models of software, models of computation platforms and models of networks. They classify microcontroller-based systems modeling approaches according to the aspect of the system displayed by the model and tasks that can be solved. This classification is as follows:

1. Models based on timed actors for timings and performance [73].
2. Event-based models for computations, communications and control [74].
3. SCADA model for load balance, stability and integrity of the system [75].
4. Ordinary differential equations and automata for non-complex systems [76].
5. Continuous-time models of dynamics for physical processes [77].
6. MDD (Model-Driven Development, [78]), MIC (Model-Integrated Computing, [79]) and DSM (Domain Specific Modeling, [80]) for software elements.
7. Multi-agent models for interaction between system elements [81].

In [82] an example of the verifying of the timing correctness and performance of the microcontroller-based system is presented. Authors are using the following verification models:

- functional relations between inputs and outputs of the system;
- timing of components;
- communication between components;
- synchronization constraints of components.

Authors perform validation in TrueTime (Matlab/Simulink), verification in UPPAAL (specification of verification models, [83]) and model checking to check whether the microcontroller-based system implements the requirements. With help of model checking, authors verify stability, safety (invariance) and reachability of the system.

In [84] a microcontroller-based systems design methodology is proposed. This methodology contains seven main steps – from system boundary definition to multi-agent and collaboration modeling:

1. *System boundary definition* is related to the black box and white box analysis. The implementation is based on SYSML (The Systems Modeling Language, [85]) diagrams or Dymola [86]/Modelica [87] models.
2. *Multi-view or multi-level modeling* is based on the MBSE approach (Model-Based Systems Engineering, [88]). The implementation is based on SYSML and OOM (Object-Oriented Modeling, [89]) for specification, analysis, design, verification and validation. Authors use SYSML for complex multi-domain system modeling in pre-design phases and for creating different diagrams that are related to specific points of view, including system behavior. They also use Simulink [90] for causal modeling and Dymola/Modelica for a casual modeling of physical processes in various physical domains.

3. *Interaction modeling* uses the port-based modeling and is related to physical support, control support and multi-domain of microcontroller-based systems. The implementation is based on SYSML and Dymola/Modelica models.
4. *Topological modeling* is based on the idea that existing scales and different viewpoints can be represented as a collection of topological entities sets and subsets linked together through different semantic degrees. The authors are using a set of directed graphs to represent the dependencies between sub-systems, components and related parameters. This is due to the fact that graph-based algorithms are good for representation of existing dependencies within the system structure and in evolving of the system boundaries. The implementation is based on directed graphs (algebraic topology, [91]).
5. *Semantic interoperability* is related to definition of existing viewpoints with help of the ontology and design knowledge and to decomposition of each design viewpoint through a graph-based topological analysis. The implementation is based on graph-based mapping ontologies [92].
6. *Multi-agent modeling* is related to the modeling of control and communication protocols (time delayed communication, interactions, changing of topology, communication network nodes and links, packet losses). The implementation is based on topological graphs and multi-agent modeling.
7. *Collaboration modeling* is related to the solving of multi-view issues and the issue of communication between agents with different ontologies. The implementation is based on OWL (Web Ontology Language, [93]).

So, their microcontroller-based system model contains: external and internal interactions, process control, behavior simulation, representation of topological relationships and interoperability through multi-agent platforms.

In [6] the authors mentioned two main challenges for microcontroller-based systems designers: nature of information and uncertainty in design. They divide the design process into functional and architectural. For architectural design authors recommend to use ADL (Architecture Description Language, [94]) which is great for dynamically modeling architectures. In their experiments, authors were using Modelica for system model representation and transferring it to the mathematical model for simulation. Authors are modelling microcontroller-based systems as an assembly of components and associated interfaces between them. They use continuous models for dynamics of the physical components and a discrete model for behaviors of the computing components. The main challenge for their approach is in joining these two models to determine important functional and system parameters and future optimization.

In [95] an object-oriented workflow language for formalizing microcontroller-based systems processes within heterogeneous and dynamic environments is presented. Workflows (or processes) are used to model the high-level behavior of the system and divided on the following levels of abstraction: *process meta-meta model* — defines semantic and syntactic elements and structures; *process meta model* —

defines all elements, types, relations and their structural combinations; *process model* — defines abstract description of the process; *process instance* — defines concrete process at execution time. Each workflow contains the following parts: process step, transition, data, event, logic step, process and handling entities. The implementation is called the component-based meta model of the system and based on EMF (Eclipse Modeling Framework, [96]).

In [97] a tool for analyzing the system's security called Tamarin Prover is considered. This tool aims to automatically verify data transfer protocols in the presence of an active attacker. Moreover, the specification language of this verifier is primarily focused on the presentation of cryptographic primitives and their properties. As a rule, Tamarin Prover is used for verification of individual data transfer protocols in critical infrastructures, for which manual testing is simply not enough.

In [98] the approach for verification of timing performance of the NAS (Network Automation Systems) is presented. The response time of the verification approach consists of three main phases:

1. *Model building* is related to specification of the component reaction times and measuring of their performance.
2. *Modeling* is related to the proposition of the component-based models – network architecture and interconnections.
3. *Verification* is related to abstraction of the NAS formal models as UPPAAL timed automata with their timing interfaces (based on proposition of action patterns and their timing wrapper).

At the final step of the approach, the result formal model is used to verify the total response time of the NAS using a sub-set of timed computation tree logic (TCTL) in the UPPAAL model checker.

In [99] an automatic verifier of the data transfer protocols called ProVerif is considered. This verifier is able to identify issues associated with the incorrect operation of the authentication process, the secrecy of the transmitted data and the equivalence of the properties of the analyzed protocols for an unlimited number of sessions. This verifier supports different cryptographic primitives and is able to automatically translate investigated protocols into an abstract representation based on Horn clauses [100]. This allows one to determine whether the required security properties are satisfied. One of the disadvantages of this approach is in the difficulty and even the inability to verify the strong properties of authentication algorithms. This is due to over-approximation when managing various communication sessions. However, it is one of the most effective tools for verifying the unreachability of certain conditions that are necessary to prove safety and reliability of the system.

In [101] a tool for designing secure and reliable data transfer protocols called Active Knowledge in Security protocols (AKiSs) is considered. Inside this tool, protocol specification language is used. This language is parameterized by a first-order

sorted term signature and an equational theory. This allows formalization of algebraic properties of cryptographic primitives. AKiSs can be used for verification of the trace equivalence for determinate cryptographic protocols.

In [102] the problem of automating the security service analysis within the framework of additional restrictions is investigated. The authors presented a decision-making procedure, which answers the question whether an effective analysis of security services is possible within the assigned constraints. The proposed approach represents a partial solution to the reconciliation problem under the data inaccessibility.

Orchestration and verification of web services was explored in the Avantssar Project [103]. In this project the language was employed to specify business processes, their communications, orchestration and security verification goals. Though there is an emphasis on an incremental approach in which orchestrations satisfying a goal are model-checked until one is found that also satisfies the security goals, it is also possible to compose services directly satisfying a simple security policy [102]. There is a significant gap before that work can be applied to microcontroller-based systems as it is necessary to take into account a much richer set of constraints such as timing constraints, functionality constraints as well as numerical constraints arising from the physical nature of such systems. However, the preliminary experiments showed that existing tools can be adapted to some degree to provide automated composition of microcontroller-based systems at least for some possible constraints [104].

The current state of the art analysis showed that there are a lot of tools and approaches that can be used for design, development and verification of different aspects of microcontroller-based systems. Such techniques can be aimed at hardware and software elements [105-108]; links, interfaces and protocols [109-113]; devices [1, 60, 114, 115] and systems [84, 116]. On the other hand, in the area of Industrial Internet of Things (IIoT) many new ecosystems appeared. Let's consider them in more detail.

Google Cloud Internet of Things is an ecosystem with secure data collection, machine learning based analysis, storing and visualization [117]. In addition, through this ecosystem a microcontroller-based system can be connected to Google services and work, for example, with artificial intelligence ones. For design and development of secure gateway devices they have a special solution – Cloud IoT Device Software Development Kit (SDK) [118]. This SDK contains libraries for secure connection and management of gateway devices. Moreover, this development kit is able to work with different devices for various use cases, for example: maintenance prediction, real-time asset tracking and smart agriculture. And to make secure connection of the gateway devices even easier Google made a special cryptography chip.

ARM Platform Security Architecture (PSA) is a methodology for design of secure devices from requirements analysis to their implementation [119]. The main elements of the PSA are threat models, architecture specifications and open-source implementations. Their methodology consists of the following key phases:

1. *Analysis phase* is focused on the formation of the security requirements based on the list of the possible threats.
2. *Architect phase* is aimed at working with freely available specifications for different IoT devices.
3. *Implementation phase* is concentrated on working with open-source implementations of the firmware and APIs.
4. *Certification phase* is focused on checking the correctness of the software interaction with interfaces.

On their site they have specifications for different devices (for example, for asset tracker, smart water meter and network camera). These specifications contain threat models of the corresponding devices, their security analysis and the list of security requirements.

Kaspersky Industrial CyberSecurity is a scope of technologies and services designed to secure industrial process control level, including supervisory control and data acquisition servers, human-machine interface panels, engineering workstations, programmable logic controllers and network connections [120]. Their development process consists of the following stages:

1. Building of the threat model and risk analysis process.
2. Application of the design methodology.
3. Installation of the Kaspersky Operating System [121].
4. Configuration of the monitoring of security policies.
5. Configuration of intrusion detection systems.

The Kaspersky OS provides a possibility to isolate sensitive components of the designed system. After its configuration process, all actions that have been recognized as insecure are prohibited by default.

Microsoft Azure Internet of Things is another ecosystem for the secure connection of the gateway devices to the cloud [122]. In this solution, the security of the software part is based on the Microsoft Security Development Lifecycle (SDL) [123]. Their SDL consists of the following key phases:

1. Provision of personal training.
2. Definition of the security requirements.
3. Definition of the metrics and compliance reporting.
4. Performing threat modeling.
5. Establishment of the design requirements.
6. Definition and use of the cryptography standards.
7. Management of the third-party components.
8. Performing static and dynamic testing.
9. Performing penetration testing.

The security of the hardware part is based on the authorization in the Azure IoT Hub — a managed cloud service.

Intel Internet of Things Platform is an ecosystem that provides secure connection of Intel devices to the cloud for data collection, storage and analysis [124]. This platform is also able to work with third-party solutions based on their trust level. The security of Intel devices (gateways) is based on their own design and development techniques, which ensure the security of hardware and software components as well as security policy management.

The ecosystem from Siemens is based on MindSphere – cloud-based, open IoT operating system [125]. This solution provides secure data collection, transmission and storage in the cloud. The Siemens solution is PaaS (Platform as a Service) that is hosted in secure data centers like Amazon Web Services. Connection to the cloud is provided via different protocols: HTTPS, MQTT, S7, OPC UA, Modbus, LoRaWAN, CoAP, XMPP, 6LowPan, LWM2M, AMQP. There are two ways of connecting the gateway devices to the MindSphere: MindConnect API and MindConnect devices. MindConnect API allows custom applications to collect and upload data to the cloud on suitable devices, while devices are plug-and-play and allow one to create a direct and secure connection with the cloud.

Another possible solution for developing secure software is the solution from Cisco — Cisco Secure Development Lifecycle (Cisco SDL) [126]. The company's approach consists of six sequential phases: product security requirements, third-party security, secure design, secure coding, static analysis, vulnerability testing. From the point of view of the development of a secure microcontroller-based system the most important phases are: product security requirements and secure design. So, on the secure requirement phase gap analysis is done, whose main task is to identify the necessary changes in the system to achieve the safe state. And in the phase of the secure design the process of threat modeling is done to make assumptions for possible threats and ways to mitigate them. In addition, one of the interesting features of Cisco SDL compared to Microsoft SDL is a third-party security phase, aimed at identifying possible threats from third-party software, as well as to ensure registration and timely updates of this software.

One possible approach for designing secure embedded devices is presented in papers [127, 128]. The essence of the techniques proposed in these papers is to identify and account the list of possible harmful effects, to which the microcontroller-based device may be subject in accordance with the selected model of the intruder, and also by used hardware and software components, already in the design phase. In this approach, the protection tools are a direct part of the device, ensuring its security. Let’s consider the main phases in more detail:

1. definition of functional requirements;
2. definition of non-functional requirements;

3. identifying the set of alternatives of component composition of the device in accordance with the functional requirements;
4. the choice of the optimal component composition of the device from the point of view of non-functional requirements;
5. identification of the list of possible harmful impacts on the device based on the static testing.

Thus, if the security level of the microcontroller-based device is sufficient, one can proceed to the stage of direct development. Otherwise, one should return to the first step and review the functional requirements. Unfortunately, a system based on the interaction of devices, each of which is designed in accordance with the suggested methodology, cannot be considered secure due to unique emergent properties occurring during operation of the system.

As an example of the approach for designing secure communication between microcontroller-based devices, let's consider an approach suggested in the framework of the European Community research project SecFutur [129]. The key idea of the solution, proposed in this project, was to use a topological approach to build secure communication channels between devices. The task was solved by calculating the security index for the path between two points of the network graph on the basis of the numerical security values assigned to the nodes [130]. This index served as a basis for changing the requirements for devices. One of the disadvantages of the proposed approach is the fact that this approach does not take into account the interaction of devices with external systems. This limitation restricts the scope of this approach when designing secure data transfer environments.

In addition, particular solutions, which adapt secure data transfer protocols to apply them for communication of microcontroller-based devices are widely available. The need for such adaptation is stipulated by several reasons: the limited computing resources of such devices, the amount of payload available for transmission in the data channel, and the ability to store relatively small amounts of data. Let's consider some examples.

Implementation of the VPN communication for microcontroller devices is presented in [131]. The developed solution is based on the adaptation of IPsec. This solution requires 8 kB of RAM and 64 kB of microcontroller memory, which gives the possibility for using IPsec in the interaction of medium-power devices.

A review of cryptographic solutions for microcontroller-based devices is presented in [132]. It is shown that despite the presence of strong limitations on energy efficiency, computational capabilities, the amount of stored data and uncontrolled interaction in an unreliable data transfer environment, the effective protection can be built by the proper selection of cryptographic algorithms, their parameters, as well as optimization and the use of low-power solutions.

The general issue of most solutions is that they are focused on the certain aspects of the security, which ensures their inapplicability for providing the security of microcontroller-based systems in general. For example, techniques for software do not take into account that the functionality of individual components of such systems is determined not only by software, but also by hardware. Moreover, the relationship between hardware and software elements can be quite strong in microcontroller-based devices, which leads to additional restrictions that significantly affect the process of their design, development and verification.

An important drawback of the techniques for building blocks is that the designed device is viewed in isolation from the system. It means that not all security aspects would be taken into account and the security of the system as a whole will not be ensured. Also, there are extensions of these techniques that are aimed at ensuring the security of the devices and network between them. The drawback is that such techniques provide secure connection between designed systems and external systems only from the designed system side, which can lead to security issues during design of complex multi-level systems.

In addition, in the area of techniques for data transfer environment only solutions applicable within a specific platform and architecture are widely used. Such solutions are aimed at adapting secure Internet protocols for their application as part of the interaction between microcontroller-based devices. The need for such adaptations is associated with the limited computing power of such devices, the size of the payload available for transmission in the data channel, and the ability to store relatively small amounts of data on them.

According to the related work analysis, there are a number of solutions for providing the secure connection between IIoT gateways and the cloud of the related ecosystem for subsequent data analysis, its secure storage and visualization. The drawback of these solutions is in binding to the specific hardware, software, platforms and architectures. It results in the possibility to secure only individual devices that can be connected to the cloud, while communication between other devices is not taken into account and delivered to physical security. These solutions also do not take into account the optimization process of the designed system due to limitations like computational complexity, energy efficiency, size and price. It means that resulting systems may not be reasonable for a developed use case because of no trade-off between resources and security level.

The current state of the art analysis also showed that there are a lot of tools and approaches that can help to model different aspects of microcontroller-based systems: physical processes, software and hardware elements, platforms, network, timings, performance, computations, load balance, interactions, system behavior, topological relationships, interoperability, system boundaries and hierarchy, workflows and business processes. The drawback is that most of such approaches

do not take security into account. Moreover, integration of standalone solutions within a single approach is a difficult task due to their incompatibility. It means that in most situations it is difficult or even impossible to transform one particular model into another without significant losses due to the lack of necessary data.

It means that a general approach for solving the issue of secure microcontroller-based systems design is not done yet. Therefore, this work is aimed at developing an original approach for the design of microcontroller-based physical security systems. Among all possible microcontroller-based systems, in this work only physical security systems were chosen as an area of the application, because in such systems during the design process it is required to ensure not only the functionality of the system, but also to ensure its security against cyber-physical attacks. The main features of microcontroller-based physical security systems are presented in more detail in Section 1.3.

1.3. Features of the microcontroller-based physical security systems

Physical security describes security measures that are designed to deny unauthorized access to facilities, equipment and resources and to protect personnel and property from damage or harm. Physical security involves the use of multiple layers of interdependent systems that can include access control system, fire alarm system, security alarm system, closed-circuit television system, light control system, climate control system, automatic telephone system, etc.

Physical security systems are generally intended to deter potential intruders, detect intrusions, monitor intruders and trigger appropriate incident responses. It is up to security designers, architects and analysts to balance security controls against risks, taking into account the costs of the system development along with broader issues such as human rights, health and safety. For example, access security measures that are appropriate for a prison or a military object are most likely inappropriate in an office, although the principles are similar.

In this work based on the standard representation of microcontroller-based systems it was decided to divide possible elements of such systems into *components*, *controllers* and *devices* that are communicating with each other. Let’s consider them in more detail.

A component is something that can be connected to a *controller* and either send signals to it or react to signals from it. *Components* can communicate only with *controllers* they are connected to. *Components* can be represented as different sensors, receivers, transmitters, readers, motors, batteries, etc. It is important to note that in this work only ready-made *components* are considered, without taking into account components of electronic circuits like resistors, capacitors, transistors, diodes, inductors, etc.

A controller is something that can be programmed to work with *components* and other *controllers*. *Controllers* can communicate with *components* and other *controllers* that are connected to them. *Controllers* can be represented as different microcontrollers and single-board computers.

A device is something that represents a system of *controllers* and *components* that are communicating with each other inside it. *Devices* can communicate only with other *devices*. *Devices* can be represented as system servers, hubs, robots, stations, drones, terminals, etc.

This architecture allows one to describe the designed microcontroller-based system on the level of *devices* first and then look into their internal structure separately. Inside each device it is possible to describe communications between *controllers* first and only after that look into communications between *controllers* and *components*.

The summary of communication types can be presented as follows:

- controller ↔ component;
- controller ↔ controller;
- device ↔ device;
- system ↔ system.

The developed architecture of microcontroller-based systems is in Figure 8.

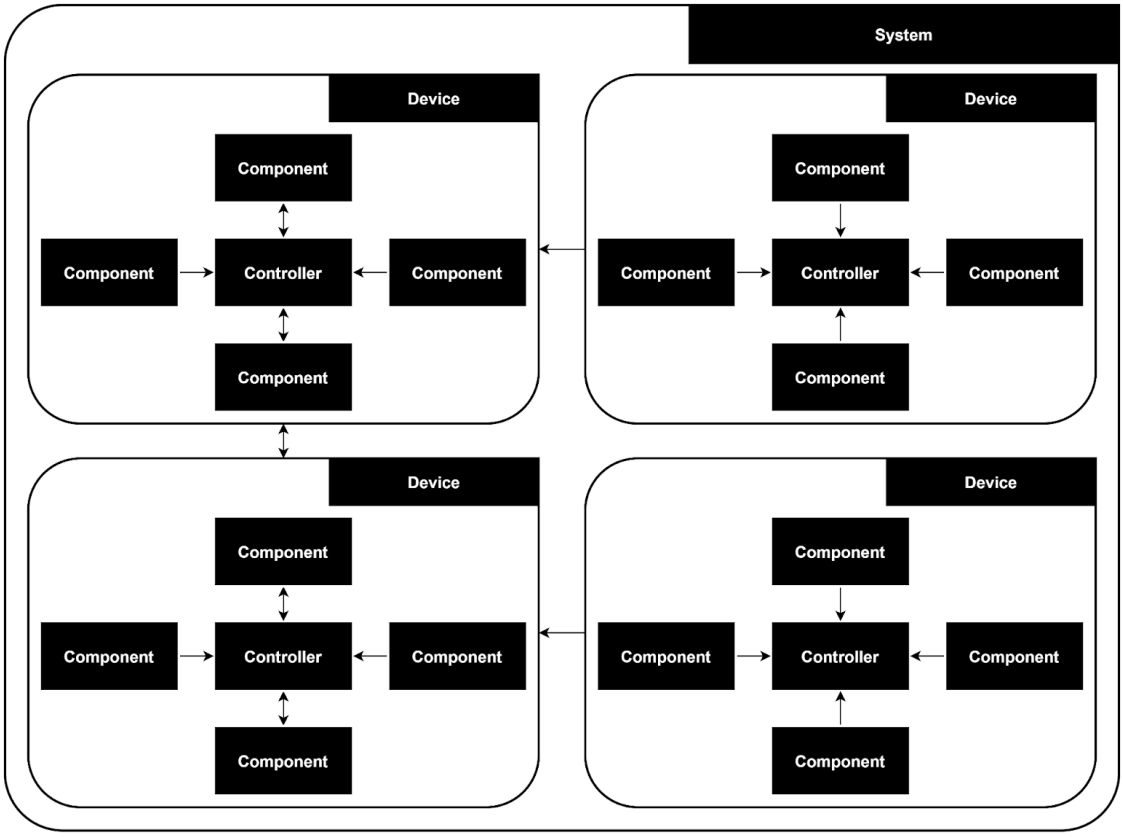

Figure 8. Architecture of microcontroller-based systems

It is important to note that the suggested architecture is focused on synthesis of the system composition, while such tasks as source code generation, case formation or development of electronic circuits are not taken into account.

One of the disadvantages of modern approaches to ensuring the security of microcontroller-based systems is the introduction of security elements after the development stage. This leads to the fact that the security system is an outer shell of the microcontroller-based system, bypassing which leaves this system defenseless against the intruder. The peculiarity of the approach proposed in this work is that instead of developing a separate security system, it is proposed to ensure that the system is secure against attacks during the design stage.

1.4. Requirements for the design methodology

To assess the results obtained in this work, an analysis of the feasibility of the requirements for the design process of microcontroller-based physical security systems was carried out. The feasibility of the requirements was validated based on the developed software prototype.

The requirements can be divided into two groups: functional and non-functional. Functional requirements are a list of functions and define the actions that a prototype must perform. Non-functional requirements describe the system requirements and constraints imposed on the resources consumed by the prototype.

Let's define a set of functional requirements for the design process of microcontroller-based physical security systems. The prototype should allow:

1. *Building an abstract representation of the designed system.* It is necessary to take into account the variety of components and their parameters, their nesting, interconnections, potential conflicts and compatibility.
2. *Finding a trade-off between the resources spent and ensuring the security of the system*. It is necessary to take into account the variety of attacker's parameters, as well as their relationship with the possibility of implementing attack actions within the framework of a specific implementation of the designed system.
3. *No restrictions on platforms and architectures of the devices to be designed*. It is necessary to have a solution that is not binded to the specific hardware, software, platforms and architectures.
4. *The extensibility of the design process*. It is necessary to have a solution that allows one to change and expand the parameters of the system elements, attacker, attack actions, methods and means of protection.
5. *Taking into account the physical layer of the designed systems.* New models and algorithms are required to correctly represent the inner interactions of microcontroller-based systems and ensure their security.

The set of non-functional requirements can be divided into the following groups: *time consumption*, *validity* and *resource consumption*. Let's consider them in more detail.

Time consumption requirement checks the ability of the design process to form a result in a given time frame, which ensures the applicability of the developed prototype in real situations. This requirement is set in the form of a certain value, the excess of which is not permissible:

- the time required for the design process of the abstract model of the system should be less than 1 second;
- the time required for the design process of the detailed model of the system should be less than 4 seconds.

For the experimental evaluation, the system of mobile robots for perimeter monitoring was designed, see Chapter 6.

Validity requirement checks the correspondence of the results of the prototype work to the real state of the system's security. In this work, the number of parameters analyzed during the design process is selected as an indicator of the validity. This requirement is set based on comparison with existing solutions. Analyzed parameters can be divided into: *levels of the system*, the security of which can be ensured; *classes of attack actions* against which the system can be protected.

Resource consumption requirement characterizes the range and amount of required software and hardware resources, spent on the design process. This requirement is also set in the form of a certain value, the excess of which is not permissible:

- the number of resources required for the design process of the system should be less than 25% of the computer resources.

For the experimental evaluation, the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM was used, see Chapter 6.

It is important to note that methodology works only with ready-made components and controllers, without taking into account elements of electronic circuits. Methodology is not generating source code of the system software and firmware. Also, the parameters of the device case, its cooling and resistance to various weather conditions are not taken into account.

It is assumed that building a methodology in accordance with these requirements will allow to reduce the number of weak places and architectural defects, thereby significantly reducing the attack surface of the microcontroller-based systems. In turn, this will reduce the security risks that can lead to financial losses, loss of time as well as the safety of people, which ensures high significance of this study.

1.5. Research problem statement

In this work research problem statement is divided into description of the input data, output data, objective function and requirements of the design methodology for microcontroller-based physical security systems. Let's consider them in more detail.

Input data of the design methodology contains:

1. *General tasks of the designed system* that are formed by the operator based on wishes of the stakeholder. Number of possible tasks as well as their values are limited by the predefined template. For more information see Chapter 4.
2. *Parameters of the attacker against which the system must be protected*. Possible values of parameters are based on the attacker's model, namely, access, knowledge and resources types. These values are limited by the predefined template. For more information see Chapter 3.
3. *Work mode*. Transition from the abstract to detailed model of the system is based on the selection of concrete components and controllers instead of abstract ones. The process of selection can be done by the operator manually or by the methodology automatically and defined by the work mode. More information can be found in Chapter 5.

It means that *input data* can be represented as follows:

$$D_I = (TK,\ ac,\ kn,\ rs,\ wm),$$

where TK – general tasks (for the prototype testing purposes are limited to 3 tasks); ac – access type (from 1 to 5); kn – knowledge type (from 1 to 4); rs – resources type (from 1 to 3); wm – work mode (0 or 1).

Output data of the design methodology contains: methodology work log, abstract model of the system, list of components and controllers that are possible for selection, detailed model of the system and methodology output log. For more information see Chapter 5. It means that *output data* can be represented as follows:

$$D_O = (wl,\ am,\ sl,\ dm,\ ol),$$

where wl – work log; am – abstract model; sl – list of selectable elements of the system; dm – detailed model; ol – output log.

The objective function of the methodology for design of microcontroller-based physical security systems is aimed at maximization of the number of parameters that are analyzed during the design process. It can be represented as follows:

$$O_F: \left| LEVELS \times ATTACKS \right| \rightarrow max,$$

where $LEVELS$— levels of the system, the security of which can be ensured (4 levels according to Section 1.3); $ATTACKS$ — classes of attack actions against which the system can be protected (4 classes according to Chapter 3).

In compliance with the requirements for *time* and *resource consumption* as well as realizability of the microcontroller-based physical security system with required level of security against attackers.

Requirement for *time consumption* can be represented as follows:

$$P_T(TIME_N \leq TIME^{ACC}) \geq P_T^{ACC},$$

where $TIME_N$ — time required to design a secure system N; P_T — probability of designing a secure system in a given time frame; $TIME^{ACC}$— acceptable time for designing a secure system (1 sec for the abstract model and 4 secs for the detailed model during the design process of the system of mobile robots for perimeter monitoring); P_T^{ACC}— acceptable probability value (0.99).

Requirement for *resource consumption* can be represented as follows:

$$P_R(RES_N \leq RES^{ACC}) \geq P_R^{ACC}$$

where P_R — probability that the resources RES_N spent on the design process of a secure system do not exceed the allowable value RES^{ACC}(0.25); P_R^{ACC}— acceptable value of probability (0.99).

Summarizing the above, in this work it is required to develop the design methodology for microcontroller-based physical security systems that based on D_I provides D_O, while $P_R \geq 0.99$, $P_T \geq 0.99$ and value of O_F exceeds values of analogs.

1.6. Conclusions on Chapter 1

The analysis and systematization of modern research in the field of information security of microcontroller-based systems have been carried out. It reveals the security of such systems from the point of view of the object of attack, the attacker, the method of attack as well as methods and means of protection. The definition of microcontroller-based systems is proposed. The classification of microcontroller-based systems is given according to such attributes as complexity, connectivity, criticality and social aspect. The classification of the attackers is given

according to such attributes as type of access, method of access, intentions, knowledge and resources is proposed. The classification of attack actions is given according to such attributes as subject, object, impact method, prerequisites and consequences is considered. A classification of methods and means of protection is given according to such attributes as the principle of work, the object of protection and the problem to be solved is proposed.

Place and role of the design techniques in ensuring the information security of microcontroller-based systems was shown. The drawbacks of existing solutions were pointed out. Their key issue is in focusing on certain aspects of security, ensuring their inapplicability for providing the security of such systems in general. For example, techniques for software do not take into account that the functionality of microcontroller-based devices is determined not only by software. An important drawback of the techniques for hardware and software-hardware is that the designed microcontroller-based device is viewed in isolation from the system. Commercial solutions are not applicable if the microcontroller-based system already contains devices whose hardware cannot be changed or the design requirements contain restrictions that do not allow the use of devices suitable for these requirements. Commercial solutions also do not take into account the optimization process of the designed system due to limitations like computational complexity, energy efficiency, size and price. It means that the resulting system may not be reasonable for a developed use case because of no trade-off between resources and security level. In addition, there are many solutions in which the security of the system is not considered or is not the main task.

It was concluded that a general approach for solving the issue of designing secure microcontroller-based systems is not done yet. Therefore, this work is aimed at developing the original approach for the design of microcontroller-based physical security systems. Among all possible systems, in this work only physical security systems were chosen as an area of the application, because in such systems during the design process it is required to ensure not only the functionality of the system, but also to ensure its security against cyber-physical attacks.

The developed architecture of microcontroller-based physical security systems contains components, controllers and devices that are communicating with each other. A component is something that can be connected to a controller and either send signals to it or react to signals from it. Components can communicate only with controllers they are connected to. A controller is something that can be programmed to work with components and other controllers. Controllers can communicate with components and other controllers that are connected to them. A device is something that represents a system of controllers and components that are communicating with each other inside it. Devices can communicate only with other devices. The summary of communication types is as follows: controller ↔ component, controller ↔ controller, device ↔ device, system ↔ system.

The requirements for the new design methodology were formulated. They are divided into two groups: functional and non-functional. Functional requirements: building an abstract representation of the designed system; finding a trade-off between the resources spent and ensuring the security of the system; no restrictions on platforms and architectures of the devices to be designed; the extensibility of the design process; and taking into account the physical layer of the designed systems. The set of non-functional requirements is divided into time consumption, validity and resource consumption. Requirement for time consumption is as follows: the time required for the design process of the abstract model of the system should be less than 1 seconds; the time required for the design process of the detailed model of the system should be less than 4 seconds. Requirement for validity is as follows: the number of parameters analyzed during the design process is greater than that of analogs. Analyzed parameters were divided into: levels of the system, the security of which can be ensured; classes of attack actions against which the system can be protected. Requirement for resource consumption: the number of resources required for the design process should be less than 25% of the computer resources.

In this work research problem statement is divided into description of the input data, output data, objective function and requirements of the design methodology for microcontroller-based physical security systems. Input data contains: general tasks of the designed system, parameters of the attacker against which the system must be protected and work mode. Output data contains: methodology work log, abstract model of the system, list of components and controllers that are possible for selection, detailed model of the system and methodology output log. The objective function of the methodology for design of microcontroller-based physical security systems is aimed at maximization of the number of parameters that are analyzed during the design process.

It is important to note that methodology works only with ready-made components and controllers, without taking into account elements of electronic circuits. Methodology is not generating source code of the system software and firmware. Also, the parameters of the device case, its cooling and resistance to various weather conditions are not taken into account.

The methods for the evaluation of the design methodology for microcontroller-based physical security systems according to time consumption, resource consumption and validity will be presented in Chapter 2.

Chapter 2. Methods for the evaluation of the design methodology for microcontroller-based physical security systems

This chapter describes methods for the evaluation of the design methodology for microcontroller-based physical security systems, namely, methods for the evaluation of time consumption, resource consumption and validity.

2.1. Method for the evaluation of time consumption

As was mentioned in Section 1.4 and Section 1.5, the *requirement for time consumption* defines the ability of the approach to design a microcontroller-based physical security system in accordance with the input data in a given time frame. This requirement can be represented as follows:

$$TIME_N \leq min_{s \in S}(TIME_N^S),$$

where $TIME_N$ — time required to design a system N using the developed methodology; S — set of design algorithms; $TIME_N^S$ — time to obtain the design result for the algorithm $s \in S$. It should be noted that the time required to design the system depends on the number of its devices, their abstract and detailed elements.

In order for the design methodology to be used in real time when building an abstract model, as well as in near real time when building a detailed model, it must design secure systems in a time not exceeding a certain boundary. Such a requirement for time consumption is set in the form:

$$P_T(TIME_N \leq TIME^{ACC}) \geq P_T^{ACC},$$

where P_T — probability of designing a secure system in a given time; $TIME^{ACC}$ — acceptable time for designing a secure system; P_T^{ACC} — acceptable probability value. Based on the results of related work analysis and series of experiments, $TIME_1^{ACC} = 1\ sec$ was chosen to design an abstract model and $TIME_2^{ACC} = 4\ secs$ was chosen to design a detailed model of the system.

Such time frames were chosen for the design process of the microcontroller-based physical security system that contains 3 types of devices, while the first type of devices consists of not less than 5 elements with sub-elements, second type — not less than 10 elements with sub-elements and third type — not less than 15 elements with sub-elements. It is important to note that this example should also take into

account links between devices and their elements, security recommendations to their implementation, requirements for links and elements as well as dependencies between them. For more detail, see Chapter 6.

It is also important to note that time required to design the microcontroller-based physical security system will be measured on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM.

Note that the time constraints depend on the acceptable response time of all stages of the methodology, which is a set of execution times for all of its algorithms. During this time, a number of queries to the database must be executed, possible attack actions, required methods and means of protection, related abstract elements and their sub-elements must be identified as well as their relations, requirements, compatibility and dependencies to form a selection process and detailed model of microcontroller-based physical security system.

Time consumption of the design process for an abstract model of the system is the sum of the time consumption of each stage of this process:

$$TIME^{AM} = T_1^{AM} + T_2^{AM} + T_3^{AM}$$

where T_1^{AM} — time of the formation of requirements for the system and its devices; T_2^{AM} — time of formation of the component composition of the abstract model; T_3^{AM} — time of formation of the components hierarchy, connections between them, requirements for them, dependencies between them and recommendations for ensuring system security after implementation.

Time consumption of the design process for a detailed model of the system is the sum of the time consumption of each stage of this process:

$$TIME^{DM} = T_1^{DM} + T_2^{DM} + T_3^{DM}$$

where T_1^{DM} — time of the formation of selection steps for links between system devices and elements of the devices; T_2^{DM} — time of the selection of detailed elements of the system; T_3^{DM} — time of the calculation of device parameters as well as insertion of information about selected elements into the abstract model.

The execution time of stages is considered as a random variable, the probability of which obeys the normal distribution law [145]. In this case, to estimate the execution time, the beta distribution law is used in the interval $[t_{min}, t_{max}]$ with the distribution density [146]:

$$f(t) = \frac{(t - t_{min})^{\alpha-1}(t_{max} - t)^{\beta-1}}{(t_{max} - t_{min})^{\alpha+\beta-1}B(\alpha, \beta)}, \; t_{min} \leq t \leq t_{max},$$
$$f(t) = 0, \; t_{max} \leq t \leq t_{min},$$

where t_{min} and t_{max} — minimum and maximum execution time, respectively; t — value which determines the execution time; $B(\alpha, \beta)$ — Euler function; $\alpha > 0$, $\beta > 0$ — beta distribution parameters.

The expected execution time of the process of designing an abstract and detailed models of a secure system and their variance are calculated using a two-score methodology [145]:

$$T_i = \frac{3T_i^{min} + 2T_i^{max}}{5}. \; \sigma^2(T_i) = 0.4(T_i^{max} - T_i^{min})^2$$

The probability that the stage execution time as a whole will not exceed the acceptable value $TIME^{ACC}$ is calculated as follows:

$$P_{NE}(TIME \leq TIME^{ACC}) = \Phi(Z)$$

where $\Phi(Z)$ — the value of the Laplace function at:

$$Z = \frac{TIME^{ACC} - \sum_{i=1}^{n} T_i}{\sqrt{\sum_{i=1}^{n} \sigma_i^2(T_i)}}$$

After that, according to the values of the Laplace function, given in a tabular form for the methodology for design of microcontroller-based physical security systems, the probability of designing abstract and detailed models in a given time is checked to conclude on their compliance with given requirements.

2.2. Method for the evaluation of resource consumption

As was mentioned in Section 1.4 and Section 1.5, the *requirement for resource consumption* characterizes the range and number of required software and hardware, the amount of required information arrays, human resources and other

resources spent on the implementation of the design process of microcontroller-based physical security systems. Requirements for resource consumption are set as follows:

$$P_R(RES_N \leq RES^{ACC}) \geq P_R^{ACC}$$

where P_R — probability that the resources RES_N spent on the design process of a secure system do not exceed the allowable value RES^{ACC}(0.25); P_R^{ACC}— acceptable value of probability (0.99).

The assessment of resource consumption can be carried out according to a number of particular indicators. Let's consider each of them in more detail.

Resource consumption when using central processing unit (CPU):

$$RES_{CPU} = \frac{Q_{CPU}^{DM}}{Q_{CPU}^{ALL}}$$

where Q_{CPU}^{DM} — central processing unit time spent on the design process of microcontroller-based physical security systems; Q_{CPU}^{ALL} — total available CPU time.

Resource consumption when using hard disk drive (HDD):

$$RES_{HDD} = \frac{Q_{HDD}^{DM}}{Q_{HDD}^{ALL}}$$

where Q_{HDD}^{DM} — HDD space used during the design process of microcontroller-based physical security systems; Q_{HDD}^{ALL} — total available HDD space.

Resource consumption when using random-access memory (RAM):

$$RES_{RAM} = \frac{Q_{RAM}^{DM}}{Q_{RAM}^{ALL}}$$

where Q_{RAM}^{DM} — RAM amount used during the design of microcontroller-based physical security systems; Q_{RAM}^{ALL} — total available amount of RAM.

The resource consumption corresponds the requirements, if all of the above indicators meet the condition $RES \leq RES^{ACC}$. To perform the design process, it is assumed that a separate computer is allocated, but part of the resources will be occupied by the operating system and other processes, therefore $RES^{ACC} = 0.25$.

It means that the design process for microcontroller-based physical security systems should take not more than 25% of the total resources. It is important to note that resources required to design the system of mobile robots for perimeter monitoring will be measured on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) CPU, 2 TB HDD and 32 GB RAM.

Note that because the CPU of the computer used contains 8 cores, it is required to measure load on each of them. The value of RES_{CPU} is calculated as average among them. And after all indicators (RES_{CPU}, RES_{HDD}, RES_{RAM}) are measured, their compliance with given requirements is concluded.

2.3. Method for the evaluation of validity

As was mentioned in Section 1.4 and Section 1.5, the *validity requirement* checks the correspondence of the results of the prototype work to the real state of the system's security. In this work, the number of parameters analyzed during the design process is selected as an indicator of the validity, namely:

- number of levels of the system, the security of which can be ensured;
- number of classes of attacks against which the system can be protected.

Requirements for these indicators are set by comparison with existing systems and can be represented as follows:

$$\begin{gathered} \left|LEVELS_N\right| \geq max\left(\left|LEVELS_{s \in S}\right|\right), \\ \left|ATTACKS_N\right| \geq max\left(\left|ATTACKS_{s \in S}\right|\right), \\ \left|LEVELS_N \times ATTACKS_N\right| > max\left(\left|LEVELS_{s \in S} \times ATTACKS_{s \in S}\right|\right), \end{gathered}$$

where $|\ |$ — denotes the number of elements in the set; N — suggested in this work design approach; S — set of design approaches with which N is compared; $\left|LEVELS_I\right|$ — number of levels of the system, the security of which can be ensured by the design approach I; $\left|ATTACKS_I\right|$ — number of classes of attack actions against which the system can be protected by the design approach I; $\left|LEVELS_I \times ATTACKS_I\right|$ — number of parameters that are analyzed during the design process by the approach I.

Thus, the methodology for the design of microcontroller-based physical security systems developed in this work should not be inferior to analogues in terms of levels of the system, the security of which can be ensured, and classes of attack actions, against which the system can be protected, as well as surpass them in the total number of analyzed parameters.

The developed methodology is compared with commercial solutions in terms of levels of the system, the security of which can be ensured and with scientific solutions in terms of classes of attack actions against which the system can be protected. For more information see Chapter 6.

Comparison of design approaches was made based on the publicly available data. For each parameter, the presence or absence of its consideration in the design process is determined. In this case, the following levels of the system are considered: communication between controllers and components, controllers, devices as well as systems. And the following classes of attack actions: on the level of components and their communication with controllers, on the level of controllers and their communication with other controllers, on the level of devices and their communication with other devices as well as on the level of the system and its communication with other systems.

2.4. Conclusions on Chapter 2

The goal of this work is to develop the design methodology that takes into account the maximum number of parameters during the design process of microcontroller-based physical security systems, while requirements for time and resource consumption are satisfied. Let's summarize the properties for the evaluation of results of this work in a single Table 1.

Table 1. Properties for the evaluation of the design methodology

	Description	Requirements
Time consumption	Probability that the approach is able to design a system in accordance with the input data in a given time frame.	$TIME_N \leq min_{s \in S}(TIME_N^S)$, $P_T(TIME_N \leq TIME^{ACC}) \geq P_T^{ACC}$
Resource consumption	Probability that the amount of used resources (CPU, HDD, RAM) will not exceed the allowable value.	$P_R(RES_N \leq RES^{ACC}) \geq P_R^{ACC}$
Validity	Number of levels of the system, the security of which can be ensured, and number of classes of attack actions against which the system can be protected.	$\lvert LEVELS_N \rvert \geq max(\lvert LEVELS_{s \in S} \rvert)$, $\lvert ATTACKS_N \rvert \geq max(\lvert ATTACKS_{s \in S} \rvert)$, $\lvert LEVELS_N \times ATTACKS_N \rvert > max(\lvert LEVELS_{s \in S} \times ATTACKS_{s \in S} \rvert)$

For the experimental evaluation of the developed design methodology, its software implementation was executed 1000 times for the system of mobile robots for perimeter monitoring on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) CPU, 2 TB HDD and 32 GB RAM to recieve average values of time and resource consumption. For more information see Chapter 5 (implementation) and Chapter 6 (evaluation).

One of the main findings of this work, namely the extendable set-based hierarchical relational model of microcontroller-based physical security systems is presented in the next chapter. This model describes the representation of microcontroller-based physical security systems that is used in the developed design approach.

Chapter 3. Extendable set-based hierarchical relational model of microcontroller-based physical security systems

This chapter describes how microcontroller-based physical security systems, attackers, attack actions and security elements are modeled in this work. Moreover, this chapter describes how developed models are connected into the extendable set-based hierarchical relational model.

3.1. Modeling of microcontroller-based physical security systems

To display various aspects of complex systems and detect the potential feasibility of various attack actions component-based, semi-natural, simulation and analytical modeling are used. Each modeling approach has its own abstraction level in the representation of the system, see Figure 9.

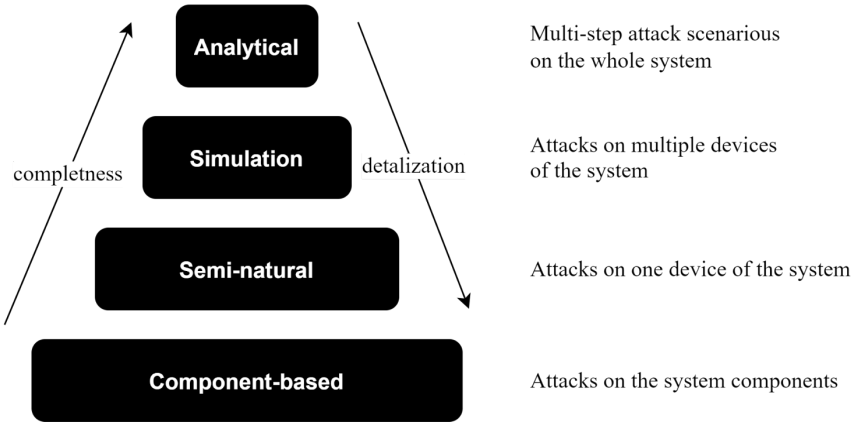

Figure 9. Comparison of modeling approaches

The component-based approach is the most detailed way to represent microcontroller-based physical security systems but it requires a lot of time and effort. Moreover, it is not possible to represent different dynamic processes with it. From the other side, with the help of analytical modeling it is possible to represent the whole system but only on a high level of abstraction. So, the performance of the solution strongly depends on the level of detailing. That is why to represent the whole lifecycle of the system, heterogeneous structures of the united models are used to overcome this issue by using different models for different cases.

For the design process of microcontroller-based physical security systems the component-based approach is the most appropriate one if it is required to take into account the security of the system as early as possible. Developed in this work model represents such systems as an extendable set-based hierarchical relational structure and consists of the following parts: building blocks (hardware and software elements), links between system elements (protocols and interfaces), an attacker and attack actions. Its overview is presented in Figure 10.

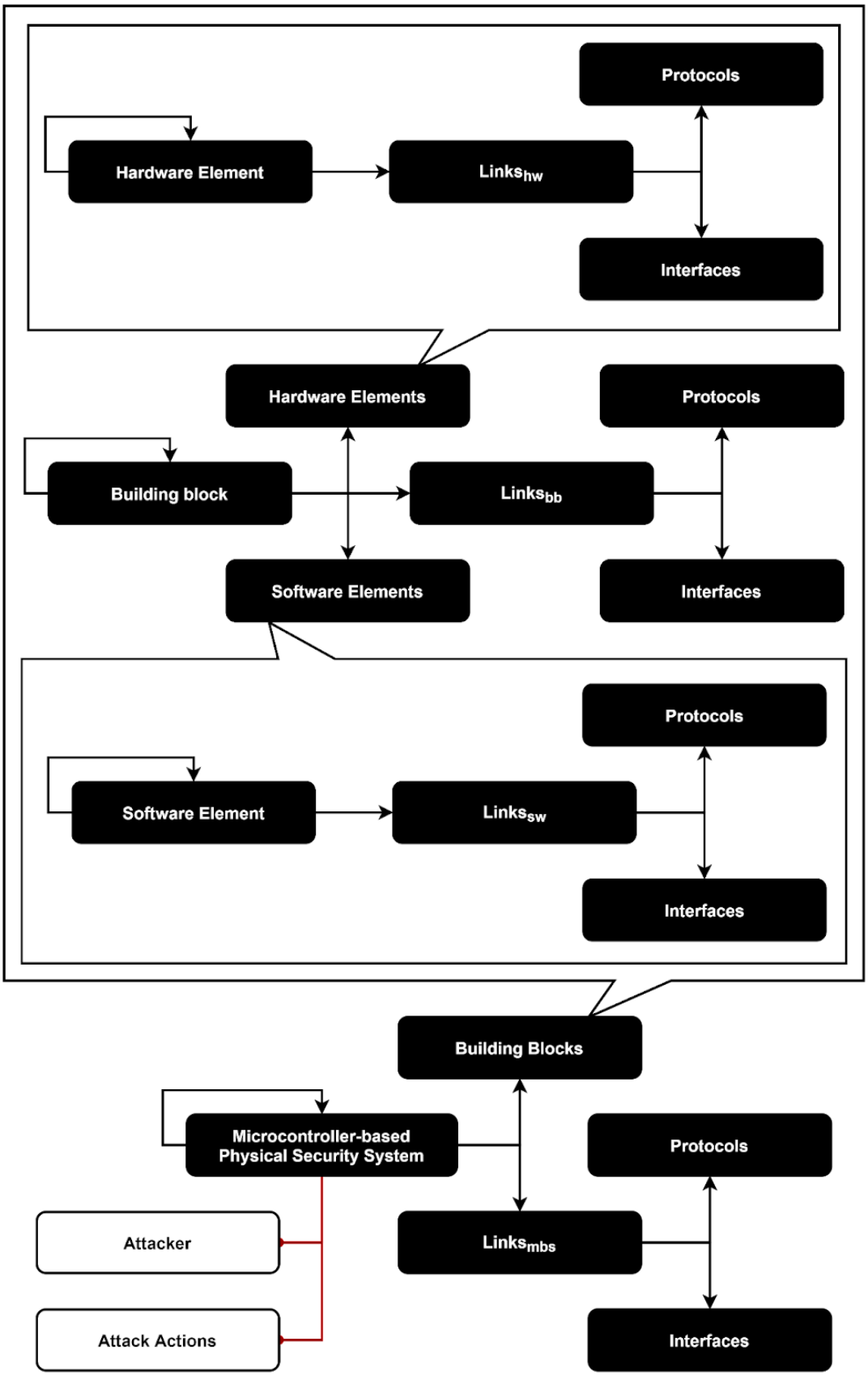

Figure 10. Extendable set-based hierarchical relational model

Black rounded rectangles are displaying the system model with its elements, while black directed arrows are displaying their hierarchy and nesting: hardware element can be a part of another hardware element or building block, microcontroller-based system may contain another system as its sub-system and so on. White rounded rectangles are displaying external models that are connected with the model of the system: attack actions impact is modeled through changes in the properties of the system or its elements while the number of possible attack actions is reduced according to the possibilities of the attacker.

One of possible ways to describe complex systems as a set of interacting building blocks is the set-theoretic approach. Let's consider it in more detail.

Any system $mbs \in MBS$ can be represented as follows:

$$mbs = (MBS', BB, L_{mbs}, a, AA, p_{mbs}),$$

where MBS' — set of microcontroller-based sub-systems of mbs; BB — set of building blocks of mbs; L_{mbs} — set of links between BB and MBS' of mbs, see Figure 11; a — attacker against mbs; AA — set of attack actions on mbs; p_{mbs} — properties of mbs.

It is important to note that each element of the system at this level is considered as an object with a set of properties and links without taking into account its internal structure. And this rule is working for sub-elements of each element as well.

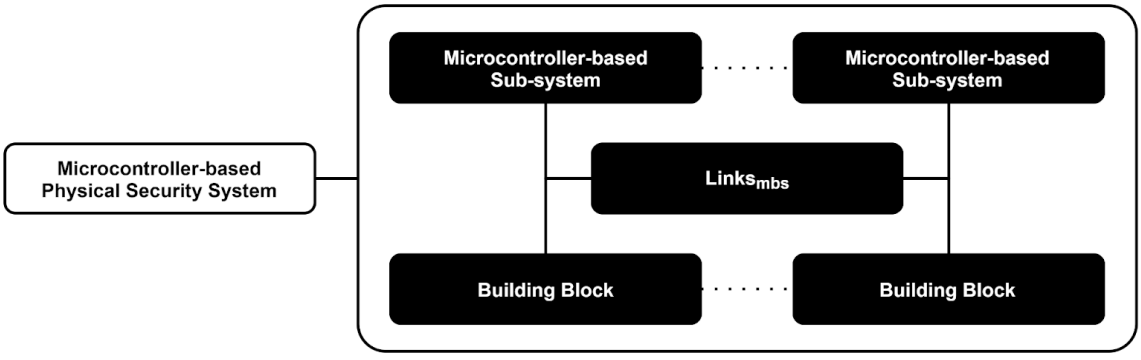

Figure 11. Links inside microcontroller-based physical security system

The model of the system allows one to represent the information about its sub-systems through MBS' and its individual blocks through BB. Information about the data transfer environment between sub-systems $mbs_i' \in MBS'$ and individual blocks $bb_j \in BB$ is represented through L_{mbs}, while properties arising from their interaction are represented through p_{mbs}.

As an example of mbs any microcontroller-based physical security system can be used: access control system, fire alarm system, security alarm system, closed-circuit television system, perimeter monitoring system, etc. The situation when mbs contains subsystems related to integrated physical security systems that combine, for example, access control, fire and security alarms systems.

A building block of mbs can be represented as follows:

$$bb = (BB', HW, SW, L_{bb}, p_{bb}),$$

where BB' — set of building sub-blocks of bb; HW — set of hardware elements of bb; SW — set of software elements of bb; L_{bb} — links between elements of bb, see Figure 12; p_{bb} – properties of bb.

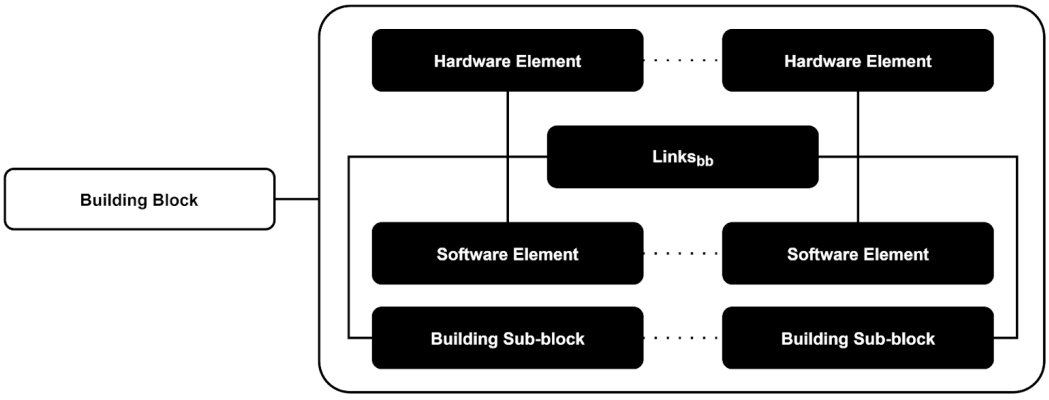

Figure 12. Links inside building blocks of the system

The model of individual blocks $bb \in BB$ allows one to represent the information about its sub-blocks through BB', hardware through HW and software through SW. Information about the data transfer environment between individual sub-blocks $bb'_i \in BB'$, hardware $hw_j \in HW$ and software $sw_k \in SW$ is represented through L_{bb}, while properties arising from their interaction are represented through p_{bb}.

As an example of a building block, any device, controller or its combination with components can be used. For example, it can be a Raspberry Pi single-board computer, micro-SD card with pre-installed operating system, ESP8266 or Iskra JS microcontroller or even server, hub, robot, station, drone, etc.

A hardware element of mbs can be represented as follows:

$$hw = (HW', L_{hw}, p_{hw}),$$

where $HW^{'}$ — set of hardware sub-elements $hw^{'}$ of hw; L_{hw} — links between elements of hw, see Figure 13; p_{hw} — properties of hw.

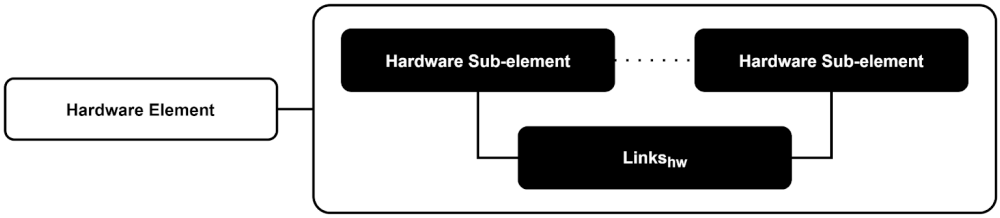

Figure 13. Links inside hardware elements of the system

The model of individual hardware elements $hw \in HW$ allows one to represent the information about its sub-elements (also hardware) through $HW^{'}$. Information about the data transfer environment between individual sub-elements $hw_{i}^{'} \in HW^{'}$ is represented through L_{hw}, while properties arising from their interaction are represented through p_{hw}.

As an example of a hardware element any component can be used: sensors, receivers, transmitters, readers, motors, batteries, etc. As an example of the hardware element that consists of multiple hardware elements, let's consider a motor shield with two collector motors that can be used for two-wheel robots. When motors are connected to the motor shield, their rotation speed and direction are controlled by its signals, while the controller of the robot can be connected to the motor shield, to control signals of the shield through the firmware.

A software element of mbs can be represented as follows:

$$sw = (SW^{'}, L_{sw}, p_{sw}),$$

where $SW^{'}$ — set of software sub-elements of sw; L_{sw} — links between elements of sw , see Figure 14; p_{sw} — properties of sw.

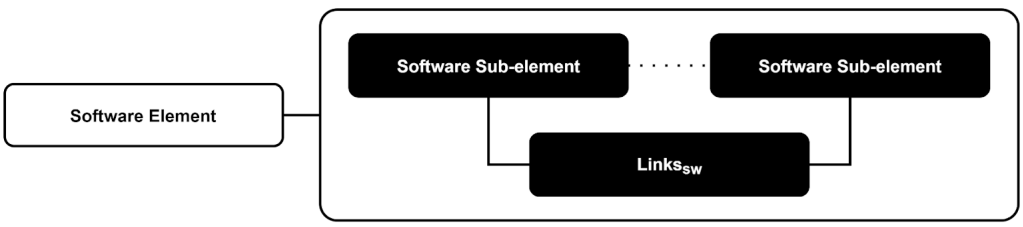

Figure 14. Links inside software elements of the system

The model of individual software elements $sw \in SW$ allows one to represent the information about its sub-elements (also software) through SW'. Information about the data transfer environment between individual sub-elements $sw'_i \in SW'$ is represented through L_{sw}, while properties arising from their interaction are represented through p_{sw}.

As an example of a software element, any algorithm, library, firmware, database, application or configuration can be used. As an example of the software element that consists of multiple software elements, let's consider a firmware of the controller that can be used as the brain of two-wheel robots. Such a firmware often contains library imports for most components that are connected to the controller as well as a lot of algorithms for the correct functioning of the robot: navigation, communication, data processing and storage, etc.

Links between elements of mbs can be represented as follows, see Figure 15:

$$L = (R, I, E, p_L),$$

where R — set of protocols that are used in L; I — set of interfaces that are used in L; E — set of communication parties of L; p_L — properties of L.

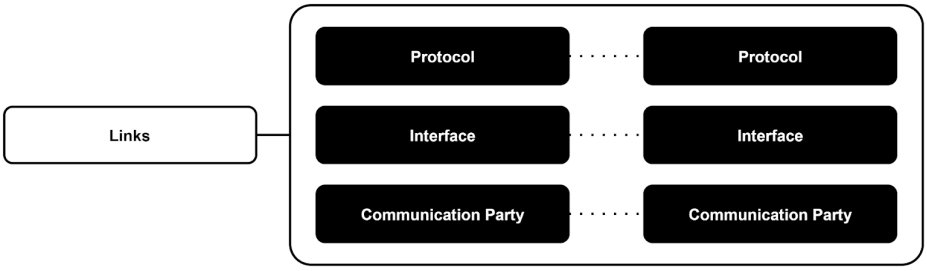

Figure 15. Links between elements of microcontroller-based systems

The model of individual links $l \in L$ allows one to represent the information about its protocols through R, interfaces through I and communication parties through E, while properties arising based on their combination are represented through p_L.

Moreover, links between elements of mbs can be divided:

- L_{mbs} — links between devices of the system;
- L_{bb} — links between controllers of devices;
- L_{hw} — links between controllers and components;
- L_{sw} — links between software elements.

It means that the model allows one to represent low level protocols between controllers and components together with connections between different algorithms inside the firmware of one of controllers, while being able to represent high level protocols between devices, see Table 2.

Table 2. Various types of links between elements

		R	I	E
L_{mbs}	Wi-Fi	IEEE 800.11	wireless 2.4 GHz	device ↔ device
	ZigBee	IEEE 802.15.4	wireless 2.4 GHz	
	Bluetooth	IEEE 802.15.1	wireless 2.4 GHz	
	nRF24L01+	ESB	wireless 2.4 GHz	
	Infrared	NEC	wireless 38 kHz	
L_{bb}	I2C	SDA + SCL	TWI	controller ↔ controller
	Serial	TxRx	UART	
	RS-232	RS232	UART	
	RS-485	RS485	UART	
L_{hw}	pin-to-pin	shared power	shield	controller ↔ component
	SVG	AVR I/O pin	three wires	
	VG	AVR I/O pin	two wires	
L_{sw}	method	functions	compiler	software ↔ software
	database	SQL queries	psycopg2	
	API	JSON structures	POST/GET	

Within the framework of the developed model, all elements are connected with each other through their properties. It means that to ensure the required level of security of the designed system, the goal of the approach is to find a reasonable combination of elements of the system according to the balance between their needs (functional requirements and non-functional limitations) and capabilities (provided functionality and resources). On the other hand, the influence of each successful attack action is represented through reduction of the system capabilities (for example, denial of service) or enhancing of its needs (for example, resource depletion).

Thus, the properties can be represented as follows, see Figure 16:

$$p = (FR, NL, PF, PR),$$

where FR — set of functional requirements (functionality that satisfaction is necessary for the element to work); NL — set of non-functional limitations (limitation that satisfaction is necessary for the element to work); PF — set of provided functionalities; PR — set of provided resources.

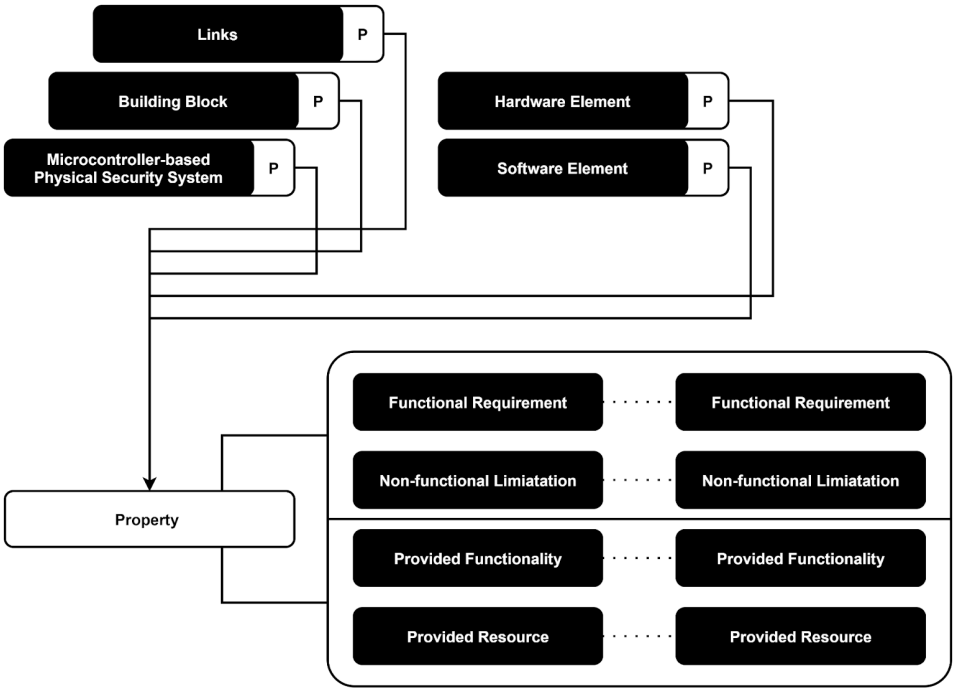

Figure 16. Properties of elements of microcontroller-based systems

The model of properties p allows one to represent the information about elements needs and capabilities through FR, NL and PF, PR accordingly. Let's consider examples of each of them in more detail.

As functional requirements of the element, any functionality necessary for it to be able to work can be used: power source, secure connection, protocol, interface, bootloader, library, operating system, compiler, driver, etc.

As non-functional limitations of the element, any limitation necessary for it to be able to work can be used: space for placement, suitable environment, voltage, current, size, volume, flash memory, digital or analog pins, disk space, ram, etc.

As provided functionality of the element, any functionality that it can provide can be used: access control, perimeter monitoring, navigation, obstacles detection, work with a component, encryption, authentication, processing, etc.

As provided resources of the element, any resource it can provide can be used: data storage, computing resources, environment for launching applications, possibility to add/remove/replace components, possibility to work with environment, etc.

3.2. Modeling of attackers, attack actions and security elements

The developed model of attacker is inspired by the classification of attackers from Section 1.1.2. According to this classification, attackers can be distinguished according to attributes such as type of access, method of access, intentions, knowledge and resources. In the developed model, only types of access, knowledge and resources are used.

An attacker against mbs can be represented as follows:

$$a = (ac, kn, rs),$$

where ac — type of access a has to mbs; kn — type of knowledge a has about mbs; rs — type of resources available to a to compromise mbs.

According to the developed model, attacker's ac can be in range between 1 and 5. This value describes the type of access an attacker has to the microcontroller-based physical security system, see Table 3.

Table 3. Attacker's types of access

	Description
1	No access to the system
2	Access to the system through global networks
3	Access to the system through local networks
4	Physical access to the system
5	Full access to the system

Attacker's kn can be in range between 1 and 4. This value describes the amount of information available to the attacker about the system, see Table 4.

Table 4. Attacker's types of knowledge

	Description
1	General knowledge about the system from publicly available sources
2	Knowledge about parameters of the system
3	Knowledge about means of protection of the system
4	Knowledge about software and hardware of the system

Attacker's rs can be in range between 1 and 3. This value describes the number of resources available to the attacker, see Table 5.

Table 5. Attacker's types of resources

	Description
1	Widely-spread software tools and known vulnerabilities
2	Specialized software tools and previously non-used vulnerabilities
3	Possibility to investigate the system

In the developed model, the structure of attacker's access, knowledge and resources types is hierarchical. It means that a_1 with $ac_{a_1} = 3$ is able to perform any attack action which is possible for a_2 with $ac_{a_2} = 2$ if $kn_{a_1} \geq kn_{a_2}$ and $rs_{a_1} \geq rs_{a_2}$. It also means that a_3 with $ac_{a_3} = 3$ is able to perform any attack action which is possible for a_1 if $kn_{a_3} \geq kn_{a_1}$ and $rs_{a_3} \geq rs_{a_1}$. But if there are $a_4 = (ac_{a_4} = 3, kn_{a_4} = 2, rs_{a_4} = 2)$ and $a_5 = (ac_{a_5} = 2, kn_{a_5} = 3, rs_{a_5} = 3)$ then a_4 will not be able to perform all attack actions that are possible for a_5 and vice versa.

The developed model of attack actions is inspired by the classification of attack actions from Section 1.1.3. According to this classification, attack actions can be distinguished according to attributes such as subject, object, impact method, prerequisites and consequences. In the developed model, only subject, object and impact method of attack are used.

An attack action on mbs can be represented as follows:

$$aa = (cl, oj, sj),$$

where cl — class of aa; oj — object of aa, helps to link aa with the target element(s) of mbs; sj — subject of aa, helps to link aa with a that is capable enough for its successful realization.

In this work, instead of separate impact methods, it was decided to use classes of attack actions, while each class contains multiple examples of methods. Classes of attack actions can be represented as follows:

$$cl = \{cn, cr, dv, st\},$$

where $cn - aa$ on the level of components and their communications with controllers; $cr - aa$ on the level of controllers and their communications with other controllers; $dv - aa$ on the level of devices and their communications with other devices; $st - aa$ on the level of the system and its communications with other systems.

Examples of attack actions on the cn level can be represented as follows:

$$cn = \{gie, bcd, rpt, rmt\},$$

where gie — generation of incorrect component events; bcd — bypassing component detection algorithms; rpt — replacement of the component; rmt — removement of the component.

Examples of attack actions on the cr level can be represented as follows:

$$cr = \{rfw, rbl, mup, imw\},$$

where rfw — replacement of the controller's firmware; rbl — reinstallation of the controller's bootloader; mup — malfunction of the controller's update system; imw — interception, modification or termination of wired communications.

Examples of attack actions on the dv level can be represented as follows:

$$dv = \{vau, cad, iec, iws\},$$

where vau — violation of the authentication system; cad — cryptographic analysis of transmitted data; iec — increased energy consumption; iws — interception, modification or termination of wireless communications.

Examples of attack actions on the st level can be represented as follows:

$$st = \{soc, pwr, web, dbd\},$$

where soc — social engineering; pwr — power failure; web — disruption of web services; dbd — disruption of database services.

As individual security element, any mean or method or protection can be used: anomaly detection algorithm, hidden placement of sensors, events correlation algorithm, vandal-proof device case, hardware authentication, firmware encryption, bootloader encryption, removement of physical update interface, strong login credentials, password policy, brute-force protection, strong encryption algorithms, secure key distribution mechanism, behavior-based anomaly detection, devices isolation/limitation, training of operators and users, etc.

As one can see, most security elements can be modeled as software or hardware elements of the system and be integrated into its building blocks, while some of them can be transferred as recommendations to the designed system implementation.

3.3. Connections between models

Let's consider how classes of attack actions are connected with parameters of attackers, see Table 6.

Table 6. Classes of attack actions and different types of attackers

			a										
			ac					kn				rs		
			1	2	3	4	5	1	2	3	4	1	2	3
cl	cn	gie			+	+	+		+	+	+			+
		bcd			+	+	+			+	+		+	+
		rpt				+	+		+	+	+	+	+	+
		rmt				+	+	+	+	+	+	+	+	+
	cr	rfw				+	+				+			+
		rbl				+	+				+			+
		mup			+	+	+			+	+		+	+
		imw				+	+	+	+	+	+	+	+	+
	dv	vau			+	+	+			+	+		+	+
		cad			+	+	+			+	+		+	+
		iec			+	+	+		+	+	+	+	+	+
		iws			+	+	+	+	+	+	+		+	+
	st	soc	+	+	+	+	+	+	+	+	+	+	+	+
		pwr				+	+	+	+	+	+	+	+	+
		web		+	+	+	+		+	+	+	+	+	+
		dbd		+	+	+	+		+	+	+	+	+	+

For example, let's consider the stakeholder that wants mbs to be secure against $a = (ac = 4,\ kn = 2,\ rs = 2)$. The gray coloring of the table cells is representing values of the attacker parameters. Connections between the possibility to implement attack actions and values of the attacker parameters are shown with "+". According to the content of the table, the designed system must be secure against: rpt, rmt, imw, iec, iws, soc, pwr, web and dbd. Those attack actions are shown with the help of bold edges of cells.

As we mentioned before, the structure of attacker's types is hierarchical. It means that an attacker with a certain access is able to perform any attack action which is possible for an attacker with the same access but with lower knowledge/resources. Such a dependence allows one to store data only about the threshold values of the types that are necessary for the successful implementation of attack actions. It is important to note that the developed model allows the use of various models of attackers and attack actions. So, the number of attacker's parameters, just like the permissible ranges of their values, can be changed. Likewise, for attack actions — another classification can be used and examples can be extended. The main thing is to preserve the hierarchical nature of the attacker's model and the relationship between his or her parameters and the possibility of implementing attack actions. In addition, let's consider how classes of attack actions are connected with security elements of microcontroller-based systems, see Table 7.

Table 7. Classes of attack actions and security elements

			Security elements
cl	cn	gie	anomaly detection algorithm, hidden placement of sensors
		bcd	events correlation algorithm, hidden placement of sensors
		rpt	vandal-proof device case, hardware authentication
		rmt	vandal-proof device case
	cr	rfw	vandal-proof device case, firmware encryption
		rbl	vandal-proof device case, bootloader encryption
		mup	vandal-proof device case, removement of physical update interface
		imw	vandal-proof device case, encryption, authentication
	dv	vau	strong login credentials, password policy, brute-force protection
		cad	strong encryption algorithms, secure key distribution mechanism
		iec	behavior-based anomaly detection, devices isolation/limitation
		iws	strong encryption algorithm on access point, strong login credentials, public key pair-based authentication
	st	soc	training of operators and users, security policy
		pwr	uninterruptible power supplies, backup power supply
		web	firewall, update mechanism, backup mechanism, logging mechanism
		dbd	input validation, strict access policy, strong login credentials, separate database users for different operations

As was mentioned in Section 1.2, possible attack actions are defined by the system elements composition and parameters of the attacker, against which the system needs to be protected. It means that if possible attack actions are known, then necessary security elements can be extracted. After that each security element can be interpreted as software (for example, anomaly detection algorithm), hardware (for example, vandal-proof device case) and recommendations (for example, training of operators and users).

Also, let's consider how classes of attack actions are connected with non-security elements of microcontroller-based systems, see Table 8.

Table 8. Classes of attack actions and non-security elements

			mbs
cl	cn	gie	sensors and receivers that react on the environment
		bcd	sensors that monitor environment
		rpt	any component
		rmt	any component
	cr	rfw	any controller with rewritable firmware
		rbl	any controller with rewritable bootloader
		mup	any controller with update system
		imw	controller ↔ controller, controller ↔ component
	dv	vau	device ↔ device, where authentication is used
		cad	device ↔ device, where encryption is used
		iec	devices with sleep mode/wireless interfaces
		iws	device ↔ device
	st	soc	any system with operators or/and users
		pwr	any system that relies on power grid
		web	any system with web-services
		dbd	any system with database

Relations between attack actions and non-security elements are defining the attack surface of the system. Understanding the attack surface allows its reduction on early stages of the system life cycle, significantly increasing its security level.

As was mentioned before, all elements of the developed model are connected through properties — their needs and possibilities. And algorithms that are used for the design of microcontroller-based physical security systems are taking it into account, see Chapter 4. But before that it is important to note another key aspect of the developed model — origin of emergent properties in the process of combining the elements of the system.

Due to the fact that collaboration requires additional resources, values of properties of elements in some cases cannot just be summarized to find out properties of their combination. To make the calculations more realistic, special modifiers that reduce values of properties are required. Their work can be represented as follows:

$$f_p(x) = p_x,\ x = (y_1, \ldots, y_n) \mid n \in N,$$

$$f_p(x) = \bigcup_{i=1}^{n} f_p(y_i) \cdot ep_x = f_p(y_1) \cdot ep_x^{y_1} \cup \ldots \cup f_p(y_n) \cdot ep_x^{y_n},$$

where x — element of mbs or mbs itself, which consists of $y_i \mid i \in 1 \ldots n$ sub-elements; p_x — properties of x; $ep_x^{y_i} \mid i \in 1 \ldots n$ — emergent properties that are related to y_i sub-element of x.

In other words, ep are considered as properties that do not correspond to individual elements. During the formation process, influence of the combination and/or interaction of individual elements is considered as emergent properties. At the same time, an emergent property that already affected FR and NFL of an individual element, when forming another element on its basis, no longer has an effect, because it has already been taken into account.

Moreover, operation $\cup$ separately combines FR, NFL, PRF and PRR. This is due to the fact that some of FR and NFL may be common for individual elements of the system while PRF and PRR of some of elements may cover individual NFL or FR of other elements. For example, fr_i related to power supply can be covered once for all connected elements, while fr_j related to the memory space can have its limits. It's even more difficult in a case when fr_q is related to resources that are not used all the time, so they can be shared between the elements.

The influence of ep can be classified by the area and result of the influence, see Figure 17. By the area of the influence ep can be divided into properties that are arising on the level of components, controllers and devices. And because ep are arising in the process of elements combining and interacting, ep on the level of the whole system are already taken into account on the level of devices and not

considered separately. By the result of the influence ep can be divided into positive, neutral and negative ones. Note, neutral ep imply their absence, but they are necessary for the completeness of the classification.

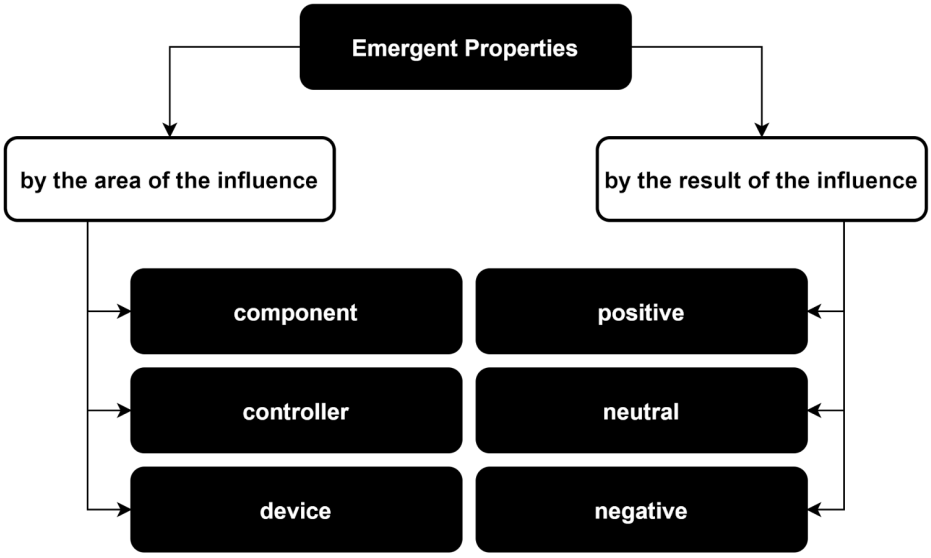

Figure 17. The classification of the emergent properties

Moreover, ep can have the area and the result of the influence at the same time, so variations like “electronic component-positive”, “microcontroller-negative” and vice versa are possible.

On the *level of components* ep are arising during the formation of controller ↔ component communications. Such formation represents a combination of hardware and software elements as well as building blocks in terms of the developed model.

For example, let’s consider the work of the controller with several MFRC522 — 13.56 MHz RFID readers. Full reading of the tag ID takes around 43 ms while only one reader can work at a time and the readers themselves do not have internal memory. It means that an RFID tag drawn faster than 43 ms past two RFID readers that are connected to one controller will be read by only one of them.

On the *level of controllers* ep are arising during the formation of controller ↔ controller communications. Such a formation represents a combination of building blocks in terms of the developed model. For example, let’s consider the interaction of controllers within the I2C bus. The I2C connection can be organized on the basis of the TWI and Wire.h library. And because of the data bus, the speed of receiving and processing messages by primary device directly depends on the number of its secondary devices as well as the number of events that would be generated by secondary devices. As experiments showed [133], it also depends on the size of the messages transmitted, the presence of confirmation of receipt, authentication of devices and encryption of the transmitted data.

On the *level of devices* ep are arising during the formation of device ↔ device communications. Such a formation represents a combination of building blocks in terms of the developed model. For example, let's consider the formation of secure communication between a single-board computer Raspberry Pi and a remote update server based on OpenVPN [134]. The bandwidth of such a connection will highly depend on the configurations on the client and server sides as well as the number of connected devices.

Positive result of influence of ep entail the fulfillment of one or more fr and/or nfl. For example, let's consider the process of the connection of controllers to the Ethernet network. As a rule, it requires the soldering of the controller to provide the necessary current and voltage, as well as interaction via certain pins for Ethernet chip and RJ45 interface. As an alternative, it is possible to use ready-made solutions like Ethernet Shield for Arduino microcontrollers.

Negative result of influence of ep entail either the termination of one or more fr/nfl or introduction of additional ones. As an example of an additional fr, let's consider the joint work of ATmega32U4 (firmware) and Atheros 9331 (Linux) processors in Arduino Yun microcontroller. To call Linux commands at the ATmega32U4 level a special interpreter is required — the Bridge.h library. In such a situation, the presence of this library becomes an additional fr which affects the nfl that are associated with firmware size and computational complexity of the solution.

3.4. Conclusions on Chapter 3

The component-based approach is the most detailed way to represent microcontroller-based physical security systems but it requires a lot of time and effort. Also, this approach is the most appropriate one if it is required to take into account the security of the system as early as possible. Developed in this work model represents such systems as an extendable set-based hierarchical relational structure and consists of the following parts: building blocks (hardware and software elements), links between system elements (protocols and interfaces), an attacker and attack actions.

Within the framework of the developed model, all elements are connected with each other through their properties. It means that to ensure the required level of security of the designed system, the goal of the approach is to find a reasonable combination of elements of the system according to the balance between their needs (functional requirements and non-functional limitations) and capabilities (provided functionality and resources). On the other hand, the influence of each successful attack action is represented through reduction of the system capabilities (for example, denial of service) or enhancing of its needs (for example, resource depletion).

The developed model of attacker is inspired by the classification of attackers from Section 1.1.2. According to this classification, attackers can be distinguished according to attributes such as type of access, method of access, intentions, knowledge and resources. In the developed model, only types of access, knowledge and resources are used. Type of access can be in range between 1 and 5, from no access to full access to the system. Type of knowledge can be in range between 1 and 4, from general knowledge to knowledge about software and hardware of the system. Type of resources can be in range between 1 and 3, from widely-spread software tools and known vulnerabilities to possibility to investigate the system.

The developed model of attack actions is inspired by the classification of attack actions from Section 1.1.3. According to this classification, attack actions can be distinguished according to attributes such as subject, object, impact method, prerequisites and consequences. In the developed model, only subject, object and impact method of attack are used. But instead of separate impact methods, it was decided to use classes of attack actions, while each class contains multiple examples of methods. Four classes of attack actions are used in the model, namely, on the level of: components and their communications with controllers; controllers and their communications with other controllers; devices and their communications with other devices; the system and its communications with other systems.

Note that the developed model allows the use of various models of attackers and attack actions. So, the number of attacker's parameters, just like the permissible ranges of their values, can be changed. Likewise, for attack actions — another classification can be used and examples can be extended. The main thing is to preserve the hierarchical nature of the attacker's model and the relationship between his or her parameters and the possibility of implementing attack actions.

As was mentioned in Section 1.2, possible attack actions are defined by the system elements composition and parameters of the attacker, against which the system needs to be protected. It means that if possible attack actions are known, then necessary security elements can be extracted. After that each security element can be interpreted as software element (for example, anomaly detection algorithm), hardware element (for example, vandal-proof device case) and implementation recommendations (for example, training of operators and users).

Due to the fact that collaboration requires additional resources, values of properties of elements in some cases cannot just be summarized to find out properties of their combination. To make the calculations more realistic, special modifiers that reduce values of properties are required — emergent properties. In the developed model, emergent properties are classified by the area and result of the influence. By the area of influence, they are divided into properties that are arising on the level of components, controllers and devices. By the result of influence, they are divided into positive, neutral and negative ones.

Unlike existing solutions, the extendable set-based hierarchical relational model represents a microcontroller-based physical security system instead of representing separate microcontroller-based devices. Such functionality neutralizes the disadvantages of analogs in terms of designing devices separately from their interaction with each other. Moreover, this model is modular, extensible and hierarchical, has a strong focus on security of the resulting solution as well as considers security elements as an integral part of the designed system. The extension of the model is possible by introduction of the new levels of abstraction. The modularity of the solution provides the possibility to change its individual parts without the need to change the model completely, for example, the parameters of the attacker's model or available classes of attacks can be updated. The hierarchical nature of the model allows decomposition from the whole system into individual elements, and composition from individual elements to the system as a whole.

Two more main findings of this work, namely the set of algorithms and the methodology for the design of microcontroller-based physical security systems are presented in the next chapter. These results are describing the process of microcontroller-based physical security systems design that is used in this work and is based on described in this chapter model.

Chapter 4. Set of algorithms and methodology for the design of microcontroller-based physical security systems

This chapter describes the set of algorithms and methodology for the design of microcontroller-based physical security systems. The set of algorithms is used within the framework of the developed methodology to design extendable set-based hierarchical relational models. This set consists of the following algorithms: algorithm for the formation of requirements to the system, algorithm for the formation of the system components composition, algorithm for the design of the abstract model of the system and algorithm for the design of the detailed model of the system.

4.1. Algorithm for the formation of requirements for the system

The algorithm for the formation of requirements for microcontroller-based physical security systems is used to extract attack actions that are possible for the attacker and a list of devices of the designed system, their links, communications, bases and requirements in accordance with the attacker's parameters and system's general tasks. This algorithm works with abstract requirements that can represent components of devices and their sub-components as well as links between devices, taking into account controllers used as the basis of the device and possible for each device types of communications that determine attack actions that are potentially dangerous for the designed devices.

As **input data**, the algorithm takes:

- *attacker's parameters:* are characterizing capabilities of the attacker in accordance with the developed model, see Chapter 3, and are represented by three parameters: access type (from 1 to 5), knowledge type (from 1 to 4) and resources type (from 1 to 3);
- *system's tasks*: are characterizing main tasks of the designed system in accordance with the wishes of the stakeholder, they are selected from a list of possible values, however, this list, unlike the attacker's parameters, does not have a limited range of acceptable values.

As **output data**, the algorithm provides:

- *attacker's actions*: the list of attack actions in accordance with the developed model, see Chapter 3, that are possible for the attacker based on the provided parameters (input data); each attack action has id, name and description;
- *security elements*: data structure for security elements that are required to prevent possible attack actions is JSON-based; each security element is represented by its unique identificator;
- *devices list*: the list of devices that are required to design in accordance with the general tasks to the system (input data), each device has name that is extracted based on system requirements;

- *devices requirements*: data structure for requirements for devices is JSON-based, while its keys are devices from the list of devices; by each device key the data about requirements for this device can be extracted; each requirement is abstract and has id and description;
- *devices communications*: data structure for devices types of communications is also JSON-based, while its keys are also devices from the list of devices; by each device key the data about possible for this device types of communications in accordance with the developed architecture of microcontroller-based systems, see Section 1.3, can be extracted; each type of communications has id and name;
- *devices links*: data structure for links between devices is JSON-based, while its keys are devices from the list of devices; by each device key the data about its links with other devices can be extracted; each link has id of link type (wire, wireless, etc.), link description, ability id (based on which ability this link between devices was detected by the algorithm) and ability description.
- *devices bases*: data structure for bases of devices is JSON-based, while its keys are devices from the list of devices; bases are representing individual controllers or their combinations that are necessary for the device to work and are extracted based on abilities that are required from the device.

The work process of the algorithm is automated, the operator is required for the translation of wishes of the stakeholder into the attacker's parameters and general tasks of the system. Its overview is presented in Figure 18.

The work process of the algorithm contains **6 main stages**, namely, initialization of data structures as well as getting attack actions possible for the attacker, security elements to prevent attack actions, abilities of the designed system, requirements of the designed system and device data. Last stage is divided into **7 sub-stages**, namely, getting device name, tasks, abilities, requirements, base, types of communication and links. Let's consider them in more detail.

Stage 1: Initialization of data structures. This stage defines the data structures for storing devices and their requirements, communications, links and bases. Devices are stored as a list, while their requirements, communications, links and bases are stored as dictionaries — key-value structures.

Stage 2: Getting attack actions possible for the attacker. At this stage, data about attack actions that are possible for the attacker in accordance with his or her parameters is extracted. Possible values of parameters are predefined by the model of the attacker, see Chapter 3. Concrete values of parameters are provided as input data and selected by the operator. Each action has an id, name and description. Possible attack actions are defined by the model of attack actions, see Chapter 3. Connections between parameters and actions are also defined in Chapter 3.

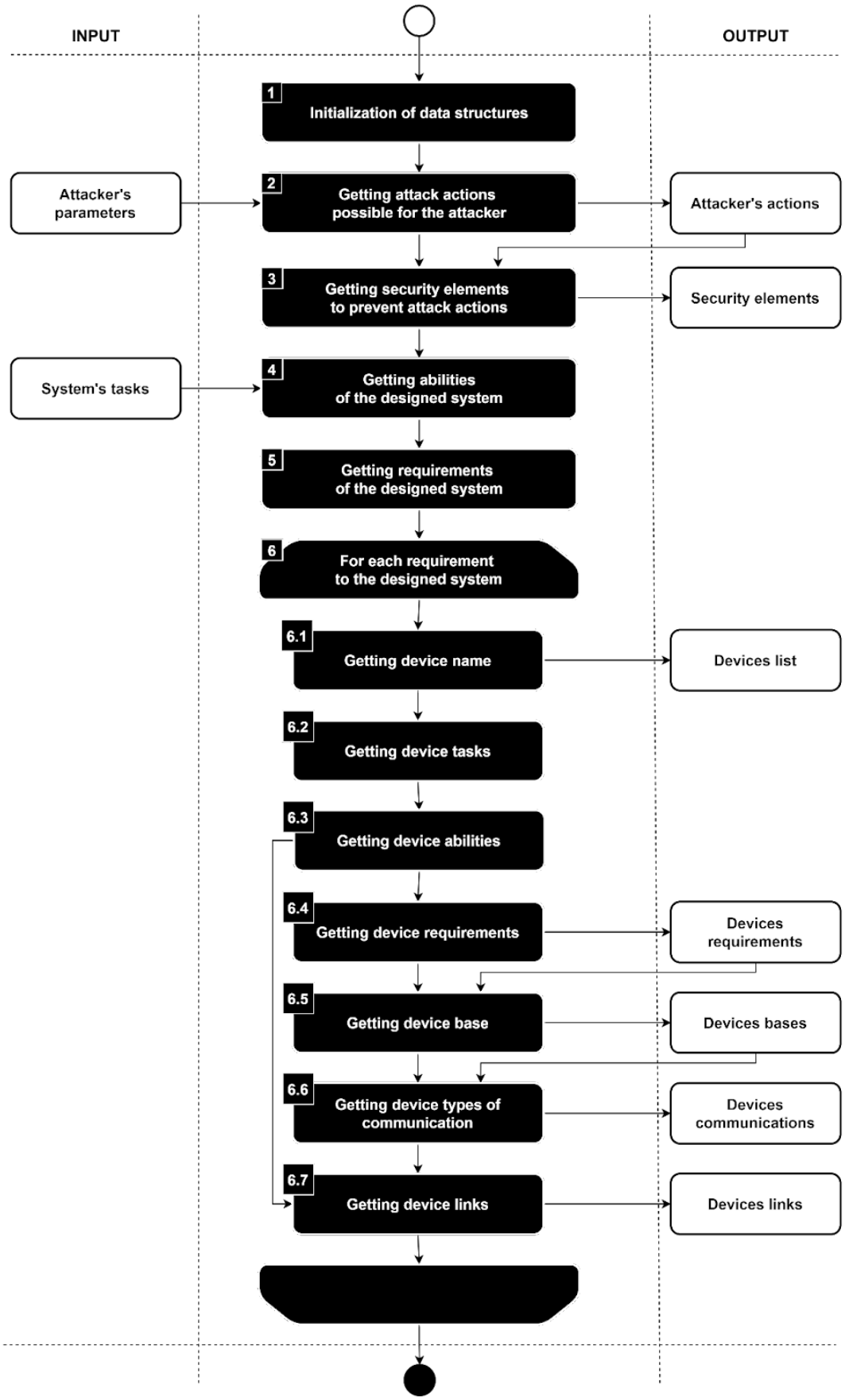

Figure 18. Overview of the algorithm for the formation of requirements

Stage 3: Getting security elements to prevent attack actions. This stage is aimed at extraction of security elements that are required to protect the system against attack actions. Actions that are possible for the attacker are provided by Stage 2. Security elements are extracted for each action separately and then combined. Each security element has an id and description.

Stage 4: Getting abilities of the designed system. At this stage, data about abilities that are expected from the designed system is extracted. Abilities are extracted in accordance with the tasks of the designed system that are provided as an input data, see Figure 19. Abilities can be interpreted as something that the designed system must be able to do to solve tasks. For example, the task “static perimeter monitoring” can be connected with the following abilities: “to communicate with mobile robots of the system”, “to provide wireless charging”, “to monitor the perimeter nearby”, and “to communicate with the server of the system”. Tasks are selected by the operator. Possible values of tasks and abilities are predefined in the database. Each task as well as ability has an id and description.

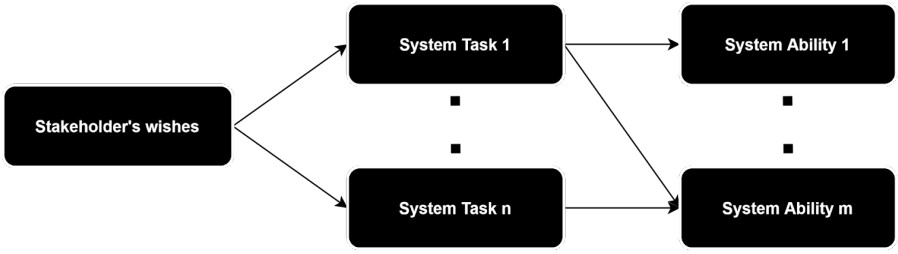

Figure 19. Connections between system tasks and abilities

Stage 5: Getting requirements of the designed system. This stage is aimed at extraction of data about requirements for the designed system. Requirements are extracted in accordance with the abilities of the designed system that are provided by Stage 4, see Figure 20. Requirements can be interpreted as something that is required for the designed system to have abilities. For example, the ability “to provide wireless charging” can be connected with the requirement “device that represents the charging stations of the system”. Possible values of requirements are predefined in the database. Each requirement has an id and description.

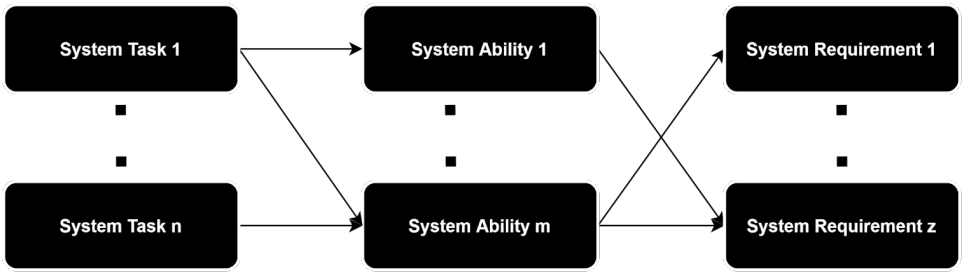

Figure 20. Connections between system abilities and requirements

Stage 6 is called getting device data. At this stage, data about devices is extracted based on requirements of the system that are provided by Stage 5. It is done for each requirement separately. Let's consider it in more detail.

Stage 6.1: Getting device name. This stage is aimed at extraction of data about the name of the device based on the provided requirement for the designed system. The name of the device is based on the requirement's description. For example, the requirement "device that represents the charging stations of the system" is transformed into "charging station". Such a transformation is possible because of the description format "device that represents the [device name] of the system". Extracted names of devices are stored in the devices list.

Stage 6.2: Getting device tasks. At this stage, data about tasks that are expected from devices of the system is extracted. Tasks of devices are extracted in accordance with requirements for the system that are provided by Stage 6.1, see Figure 21. Those tasks can be interpreted as functionality that the designed device must have to fulfill system requirements. For example, the system requirement "device that represents the charging stations of the system" can be connected with the following tasks of the device: "work cycle support", "interaction with intruders", "interaction with mobile robots", "interaction with the server". Possible values of tasks are predefined in the database. Each task has an id and description.

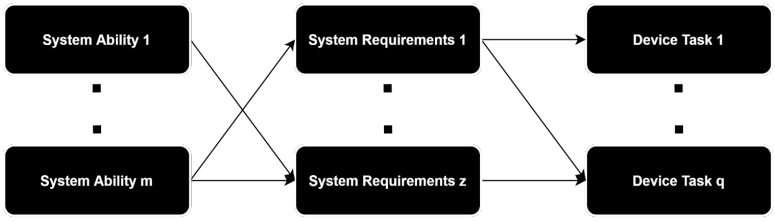

Figure 21. Connections between system requirements and devices tasks

Stage 6.3: Getting device abilities. This stage is aimed at extraction of data about abilities of the designed devices. Abilities are extracted in accordance with the tasks of the designed devices that are provided by Stage 6.2, see Figure 22. Abilities can be interpreted as something that the designed devices must be able to do to solve their tasks. For example, the task "interaction with intruders" can be connected with abilities "to detect intruders" and "to chase intruders". Possible values of abilities of devices are predefined in the database. Each ability has an id and description.

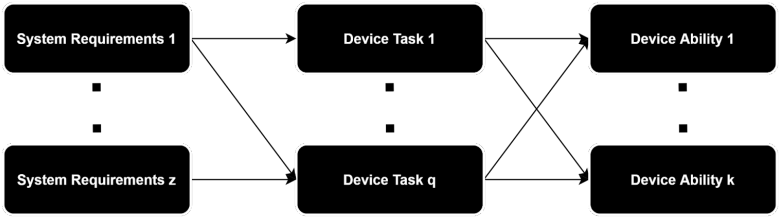

Figure 22. Connections between devices tasks and abilities

Stage 6.4: Getting device requirements. At this stage, data about requirements for the designed devices is extracted. Requirements are extracted in accordance with the abilities of the designed devices that are provided by Stage 6.3, see Figure 23. Requirements can be interpreted as something that is required for the designed devices to have their abilities. For example, the ability “to detect intruders” can be connected with the following requirements: “motion sensor”, “servo drive”, “noise sensor” and “detection algorithm”. Possible values of requirements are predefined in the database. Each requirement has an id and description.

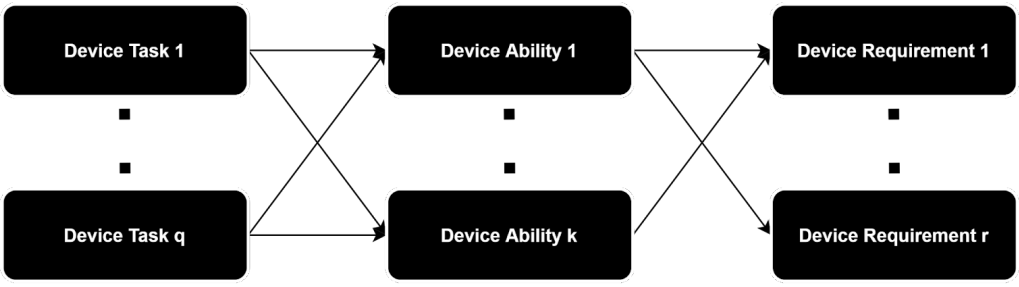

Figure 23. Connections between devices abilities and requirements

Stage 6.5: Getting device base. This stage is aimed at extraction of data about bases of the designed devices. Bases are extracted in accordance with the requirements for devices that are provided by Stage 6.4. The base of the device can be interpreted as something that represents its main computing unit. In this work, the base can have the following values: “single-board computer”, “connected microcontrollers” or “microcontroller”. Such values were selected to represent controllers of microcontroller-based devices. Extraction of the base for each device is based on all its requirements, where the necessary base is selected according to the principle of minimum allowable computing power. Possible values of bases of devices are predefined in the database. Each base has an id and description.

Stage 6.6: Getting device types of communications. At this stage, data about types of communications that are possible for the designed devices is extracted. Types of communications are extracted in accordance with the bases of devices that are provided by Stage 6.5. Types of communications can be interpreted as levels of communications that were presented in Section 1.3 during presentation of the architecture of microcontroller-based systems. Possible values of types are predefined in the database. Each type has an id and description.

Stage 6.7: Getting device links. This stage is aimed at extraction of data about links between the designed devices. Links are extracted in accordance with the abilities of devices that are provided by Stage 6.3. Links between devices can be interpreted as links of the extendable set-based hierarchical relational model that were presented in Chapter 3. Possible values of links are predefined in the database. Each link has an id and description.

As was mentioned, during most of the stages the algorithm relies on the content of the database for making decisions. The possible structure of such a database is presented in detail in Chapter 5.

This algorithm can be used to extract attack actions that are possible for the attacker as well as the list of devices of the designed system, their links, communications, bases and requirements in accordance with the attacker's parameters and system's general tasks. The output data is well-structured, while the algorithm takes into account dependencies between stakeholder's wishes and system tasks, system tasks and system abilities, system abilities and system requirements, system requirements and devices tasks, devices tasks and devices abilities, devices abilities and devices requirements, see Figure 24.

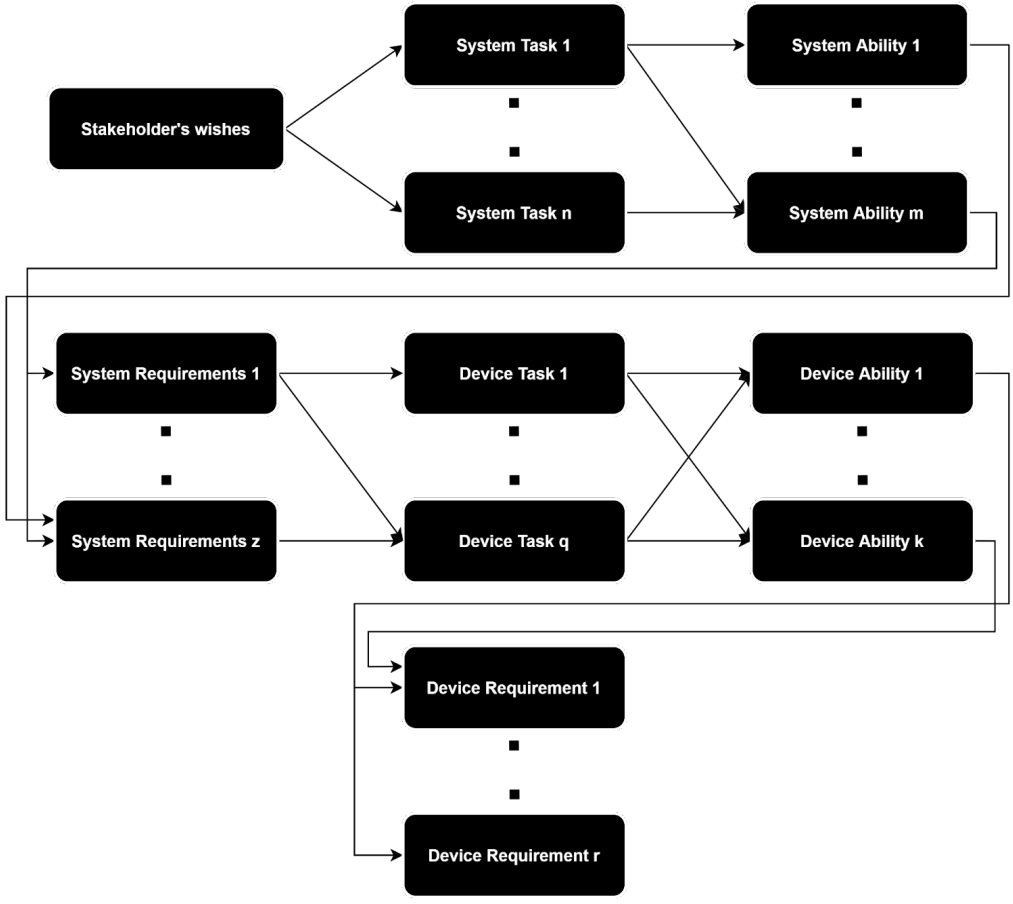

Figure 24. Connections between tasks, abilities and requirements

It is important to note that this algorithm can be useful to an expert in the design of secure systems, but its full potential is revealed only when interacting with other algorithms from this chapter within the framework of the design methodology from Section 4.5. In this methodology, this algorithm is providing the input data — attacker's actions, security elements, devices list, devices requirements, devices communications, devices links, devices bases — that is used by other algorithms, one of which is presented in the following section.

4.2. Algorithm for the formation of the system component composition

The algorithm for the formation of the microcontroller-based physical security system component composition is used to extract abstract elements and sub-elements of the devices of the system, security recommendations to the system and its devices implementation as well as abstract links between devices with related to them abilities based on attack actions that are possible for the attacker, list of devices of the system, their bases, types of communications and links, requirements for them. This algorithm works with abstract elements, links and recommendations and represents the designed system components compositions as multiple devices, each of which has multiple abstract elements, while each abstract element can have multiple abstract sub-elements. Wherein abstract elements and sub-elements are representing controllers and components as well their software, including those that are related to security.

As **input data**, the algorithm takes:

- *devices list:* the list of devices that are required to design, which is provided by the previous algorithm, see Section 4.1;
- *devices bases*: data structure for bases of devices is JSON-based, while its keys are devices from the list of devices; bases are provided by the previous algorithm, see Section 4.1;
- *devices requirements*: data structure for requirements for devices is JSON-based, while its keys are devices from the list of devices; by each device key the data about requirements for this device can be extracted; each requirement is abstract and has id and description; requirements are provided by the previous algorithm, see Section 4.1;
- *attacker's actions*: the list of attack actions that are possible for the attacker; each attack action has id, name and description; attack actions are provided by the previous algorithm, see Section 4.1;
- *devices communications*: data structure for devices types of communications is also JSON-based, while its keys are also devices from the list of devices; by each device key the data about possible types of communications can be extracted; each type of communications has id and name; devices types of communications are provided by the previous algorithm, see Section 4.1;
- *devices links*: data structure for links between devices is JSON-based, while its keys are devices from the list of devices; by each device key the data about its links with other devices can be extracted; each link has its type and description as well as ability id and description; devices links are provided by the previous algorithm, see Section 4.1.

As **output data**, the algorithm provides:

- *abstract elements and sub-elements*: abstract component composition of the system devices, where abstract elements are extracted based on requirements for the device and possible attack actions and represent

controllers, components, software and firmware, while abstract sub-elements are extracted based on abstract elements and represent algorithms, settings and requirements; data structure for abstract elements and sub-elements is JSON-based, while its keys are devices from the list of devices; by each key the data about the respective abstract elements can be extracted, while each abstract element is also a key to extract data about the set of its sub-elements; each element and sub-element has id and description.

- *security recommendations*: abstract security recommendations to the system implementation as a whole as well as for each of its devices separately that are extracted based on security elements and can't be interpreted as abstract elements or sub-elements; data structure for recommendations is also JSON-based, while it has keys for the system and all its devices; by each key the data about the respective recommendations can be extracted;
- *abstract links and abilities*: abstract types of communications that are possible between devices of the system with corresponding devices abilities that are related to their interaction; data structure for links is JSON-based, while its keys are devices from the list of devices; by each key the data about the respective links can be extracted.

The work process of the algorithm is automatic, the operator is not required. Its overview is presented in Figure 25.

The work process of the algorithm contains **2 main stages**, namely, initialization of data structures as well as getting component composition of devices. Last stage is divided into **5 sub-stages**, namely, getting abstract elements with their sub-elements, possible attack actions, additional abstract elements with their sub-elements, security recommendations to implementation as well as links between devices. Let's consider them in more detail.

Stage 1: Initialization of data structures. This stage defines the data structures for storing abstract elements and sub-elements of devices, security recommendations to the implementation of the system and its devices as well as abstract links between devices and abilities that are defining those links. All this data is stored as dictionaries — key-value structures.

Stage 2 is called getting component composition of devices. At this stage, based on the provided input — data devices list — component composition of each device of the system is extracted. Let's consider it in more detail.

Stage 2.1: Getting abstract elements with their sub-elements. This stage is aimed at extraction of data about abstract elements of devices of the system as well as their sub-elements based on provided requirements for devices and their bases. Elements are extracted recursively based on:

- provided device base;

- provided requirements for the device;
- already extracted elements.

Possible values of elements and their sub-elements are predefined in the database. Each element as well as sub-element has an id and description.

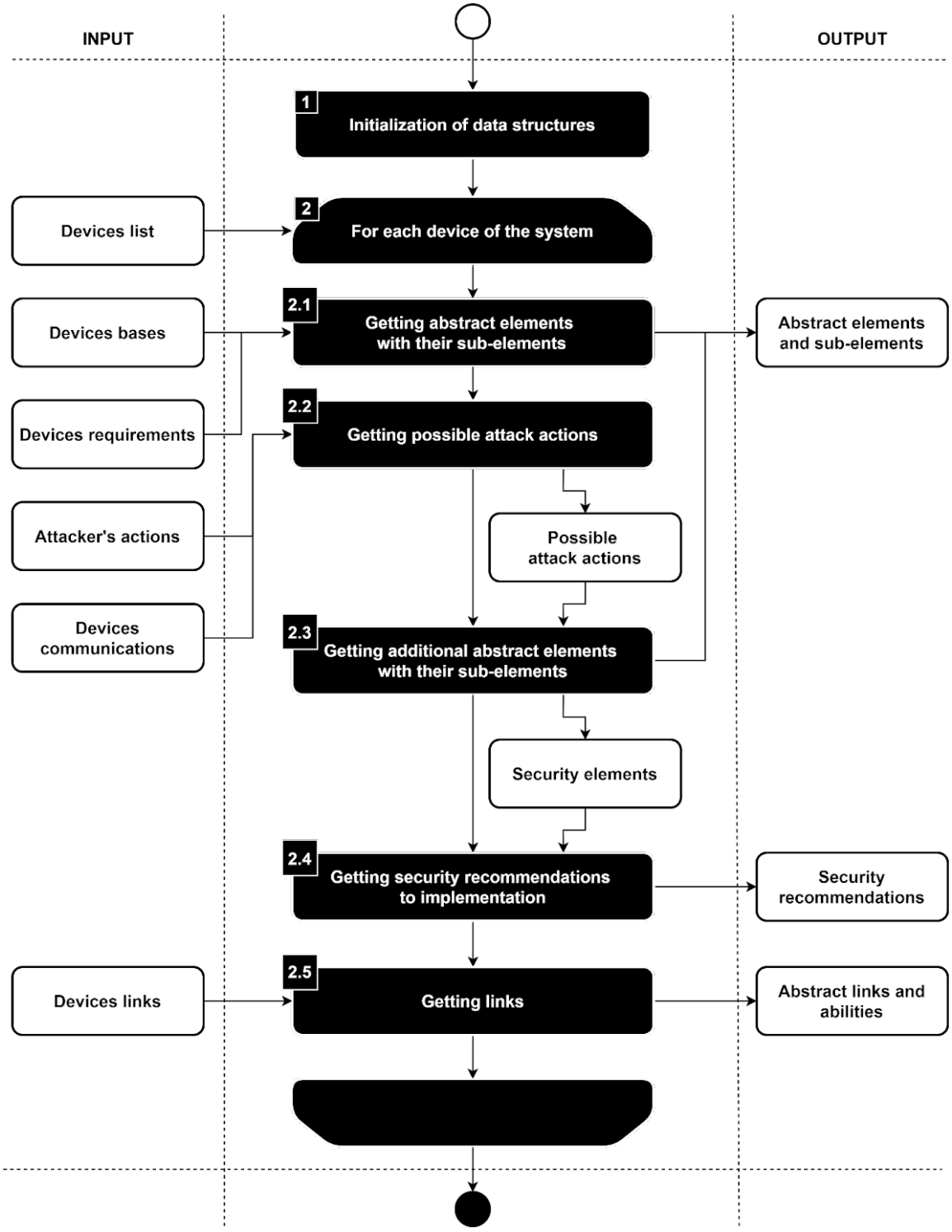

Figure 25. Overview of the algorithm for the formation of components composition

Stage 2.2: Getting possible attack actions. At this stage, data about attack actions that are possible for the designed devices in accordance with its types of communications and component composition are extracted. Types of communications that are possible for the device are provided as input data. After attack actions that are possible based on component composition and communications of the device are extracted, they are compared with attack actions that are possible in accordance with parameters of the attacker. Intersection of these two sets of attack actions allows one to get the set of attack actions that are possible on the designed device. Each attack action has an id, name and description. Possible values of actions are defined by the model of attack actions, see Chapter 3. Connections between elements and actions are also defined in Chapter 3.

Stage 2.3: Getting additional abstract elements with their sub-elements. This stage is aimed at extraction of data about additional elements and sub-elements of the device based on the provided attack actions. Additional elements are related to means and methods of protection that are necessary to prevent attack actions. Firstly, the list of required security elements is extracted. Connections between security elements and attack actions are defined in Chapter 3. After that, abstract elements and sub-elements that are representing security elements are extracted. In the end, additional elements of the device are combined with its other elements that were extracted on Stage 2.1. Once again, possible values of elements and their sub-elements are predefined in the database. Each element as well as sub-element has an id and description.

Stage 2.4: Getting security recommendations to implementation. At this stage, data about security recommendations to the implementation of the system and its devices in accordance with security elements of devices is extracted. Firstly, data about recommendations to each device implementation is extracted. After that, data about recommendations to the system implementation is extracted. Recommendation can be interpreted as a security requirement that can't be satisfied on the component composition level, that is why it can be satisfied only after implementation. For example, a recommendation to the system can be formulated as follows: "to educate operators and users of the system about social engineering attacks". Connections between security elements and recommendations are stored in the database. Each recommendation has an id and description.

Stage 2.5: Getting links. This stage is aimed at extraction of data about links between devices of the system based on the provided input data — devices links. This stage is related to transformation of the input data into another data structure called abstract links and abilities. The new data structure is JSON-based, while keys are devices from the list of devices and values are links between devices of the system. Each link has link id, link type, ability id and ability description. Possible values of ids, types and descriptions are predefined in the database.

Once again, as was mentioned, during most of the stages the algorithm relies on the content of the database for making decisions. The possible structure of such a database is presented in detail in Chapter 5.

This algorithm can be used to extract abstract elements and sub-elements of the designed system devices, security recommendations to the implementation of the system and its devices as well abstract links between devices in accordance with attack actions that are possible for the attacker, list of devices of the system, their bases, requirements, communications and links. The output data is well-structured, while the algorithm takes into account the iterative retrieval process of abstract elements of devices together with their sub-elements. At the beginning, abstract elements and sub-elements are retrieved in accordance with bases of devices, then on the basis of their requirements, after that in accordance with already extracted elements and sub-elements as well as required methods and means of protection.

It is important to note that this algorithm can be useful to an expert in the design of secure systems, but its full potential is revealed only when interacting with other algorithms from this chapter within the framework of the design methodology from Section 4.5. In this methodology, this algorithm is providing the input data — abstract elements and sub-elements, security recommendations, abstract links and abilities — that is used by the algorithm presented in the following section.

4.3. Algorithm for the design of the abstract model of the system

The algorithm for the design of abstract models of microcontroller-based physical security systems is used to construct an abstract representation of a secure system based on its devices list, their abilities, elements and sub-elements as well as security recommendations. This algorithm represents the system as an abstract hierarchical model that takes into account connections between system devices, their elemental composition, dependencies between device elements and requirements for them.

As **input data**, the algorithm takes:

- *security recommendations:* abstract security recommendations to the system implementation as a whole as well as for each of its devices separately that are extracted based on security elements and can't be interpreted as abstract elements or sub-elements; recommendations are provided by the previous algorithm, see Section 4.2;
- *abstract elements and sub-elements:* abstract component composition of the system devices, where abstract elements are extracted based on requirements for the device and possible attack actions and represent controllers, components, software and firmware, while abstract sub-elements are extracted based on abstract elements and represent algorithms, settings and requirements; abstract elements and sub-elements are provided by the previous algorithm, see Section 4.2;
- *abstract links and abilities:* abstract types of communications that are possible between devices of the system with corresponding devices abilities that are related to their interaction; abstract links and abilities are provided by the previous algorithm, see Section 4.2;
- *security elements:* abstract methods and means of protection that are required to make the designed system secure against attackers with certain parameters, interpretable as security recommendations, abstract elements and sub-elements; security elements are provided by the previous algorithm, see Section 4.1.

As **output data**, the algorithm provides the *abstract system model* that contains abstract system representation. The structure of the abstract model of the system is JSON-based and contains the following fields:

- *devices*: data about each device of the system, including its unique key, id, name, components and recommendations;
- *recommendations*: data about recommendations to the implementation of the system to ensure its security against attackers with certain parameters, including unique key, id and name (description);
- *links*: data about links between devices of the system, including its unique key, id, type, parties, dependencies and requirements.

Each element from the “components” field has its unique key and id as well as data about its own components (sub-elements), links, requirements and dependencies.

The work process of the algorithm is automatic, the operator is not required. Its overview is presented in Figure 26.

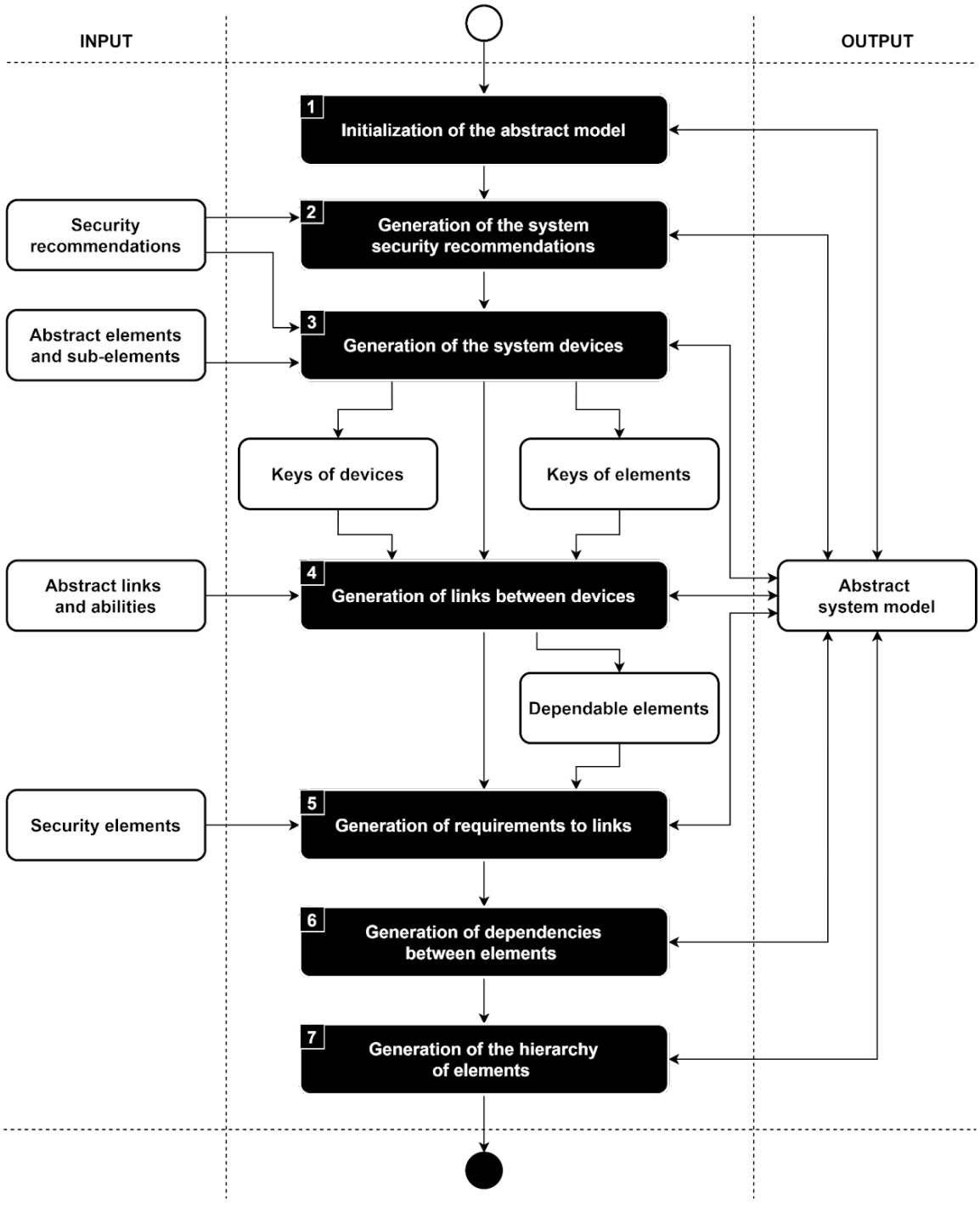

Figure 26. Overview of the algorithm for the design of abstract models

The work process of the algorithm contains **7 main stages**, namely, abstract model initialization as well as generation of system recommendations, devices, links requirements, dependencies and hierarchy. Let's consider them in more detail.

Stage 1: Initialization of the abstract model. This stage defines the data structure for storing the abstract model of the system. At the end of the stage, the abstract model consists of fields for data about devices, links between them and security recommendations for the implementation of the system.

Stage 2: Generation of the system security recommendations. At this stage, the abstract model of the system is filled with data on the recommendations for the implementation of the system related to ensuring its security. Each of the recommendations has a unique key by which its id and text description are available.

Stage 3: Generation of the system devices. This stage is aimed at filling the abstract model of the system with data about its devices. For each device, data is generated about its unique identifier, name and components composition. Data on recommendations related to ensuring the security of devices after their implementation is also generated.

The main part of this stage is the generation of the device components composition. This part contains the initialization of abstract components of each device as well as the generation of their requirements based on each component sub-elements (including security ones). For example, depending on the component of the device, it is assumed how much flash memory of the firmware it needs to work correctly.

After this stage is done, each device of the abstract system model is filled with a number of elements in their "components" field. Each element represents an abstract component of the microcontroller-based system — operating system, firmware, sensor, receiver, transmitter, database, microcontroller, etc. Each element in the abstract model has its own key that is unique only inside each device. By using this key, the data about its unique identifier, name, components, links and requirements can be extracted. It is important to note that data about each element's components and links during this stage is empty and would be filled only during stage 7.

Stage 4: Generation of links between devices. At this stage, the abstract model is filled with data on links between devices of the system. Firstly, the algorithm detects all links that are possible between each pair of devices according to their abilities. And if the link is detected, its generation starts. In the abstract model, each link has its own unique key, by which data about its unique identifier, type, parties, dependencies and requirements can be extracted. For example, the "dependencies" field is filled with data about abstract elements, the selection of a specific implementation of which directly depends on the selection of the interface and protocol of this link between devices. As an output of this stage, unique keys of links

with unique identifiers of elements the selection of which depends on the selection of a specific interface and protocol of the link are provided.

Stage 5: Generation of requirements for links. This stage is aimed at filling the abstract model with data about requirements for links between devices of the system. This field was empty after stage 4 and now is filled with data generated based on the information about security elements that are required to design a secure system. Generated during this stage requirements define if a link is wired or wireless, transfers data, signal or charge, requires encryption and/or authentication and so on.

Stage 6: Generation of dependencies between elements. At this stage, the abstract model is filled with data on requirements for elements of devices as well as with data about dependencies between them. For example, for each microcontroller data about dependencies between their selection and the subsequent selection of sensors that will be connected to them would be generated. It is done to ensure the compatibility of the elements of the device after the transmission from the abstract model to the implementation of the system. Also, for each controller that is related to control of other components like sensors, receivers and transmitters, the number of required digital and analogue pins is calculated.

Stage 7: Generation of the hierarchy of elements. This stage is aimed at the reconstruction of the “components” field of each device of the system. The algorithm generates hierarchical elements composition instead of their enumeration. The transmission to the hierarchical structure is based on a graph representation of the components of each device of the system and recursive conversions. Firstly, graph nodes are generated based on unique identifiers and keys of elements. After that the elements of each device are checked pair by pair in terms of the possibility to connect one element to another. For example, a sensor can be connected to a controller if they are compatible, while compatibility can be checked according to their parameters. And if two elements can be connected to each other then the edge between nodes that are representing them is generated.

After the graph structure for each device is generated, the process of hierarchy building starts. Firstly, the root node of the graph is obtained based on topological sorting. After that, the child node of the lowest level of the graph is obtained together with its parent node. It is required for the algorithm to encapsulate the data about the obtained child element into the “components” field of its parent element as well as for the generation of a link between them. After it is done, the data about the encapsulated child is deleted from the abstract model (this data is in the “components” field of its parent now) and the node corresponding to this child is deleted from the graph representation of the device. This process continues until no other graph node can be deleted.

It is important to note that during stages 3, 4, 6 and 7 the algorithm relies on the database for making decisions:

- *stage 3:* to decide if the abstract element of the device is a component;
- *stage 4:* to decide if devices are linked based on their abilities; to decide what abstract elements selection will depend on the selection of interface and protocol of the abstract link between devices;
- *stage 6:* to decide if one abstract element selection will depend on the selection of another abstract element; to decide how many digital or analogue pins are required to connect the abstract element to the controller;
- *stage 7:* to decide if one abstract element is compatible with another; to decide what link will be between two abstract elements.

The possible structure of such a database is presented in detail in Chapter 5.

The abstract system model is a mapping of the extendable set-based hierarchical relational model of microcontroller-based physical security systems from Chapter 3. The mapping on the level of the system is straightforward, see Figure 27.

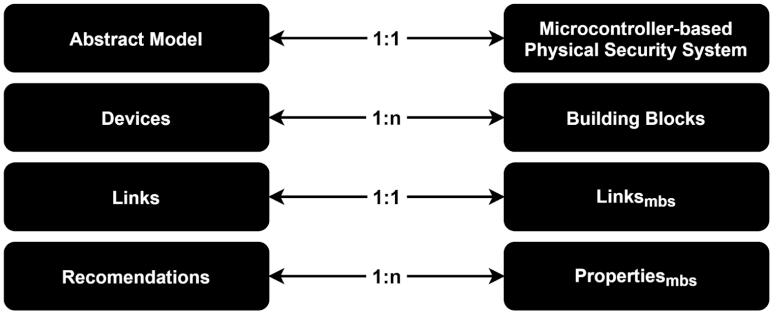

Figure 27. The mapping on the level of the system

The mapping on the level of the links between devices is presented in Figure 28.

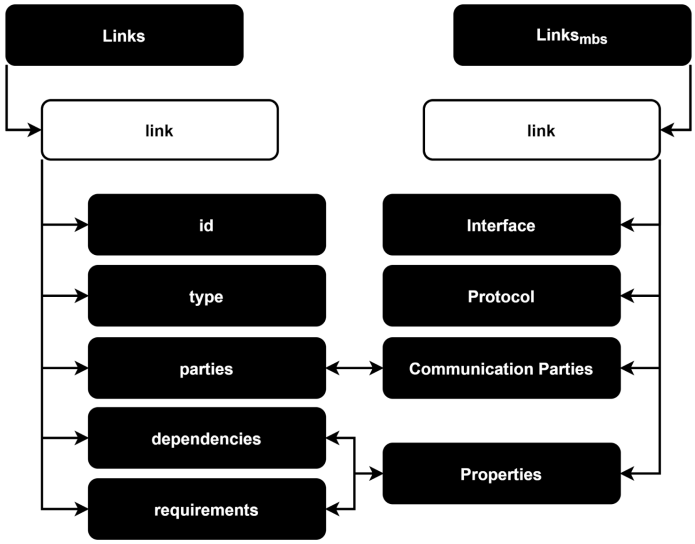

Figure 28. The mapping on the level of the links between devices

Interfaces and protocols are not mapped in the abstract model, because for such an operation it is required to select a concrete implementation of the link. It is done during the design of the detailed model and described in more detail in Section 4.4. The mapping on the level of the hardware and software is presented in Figure 29.

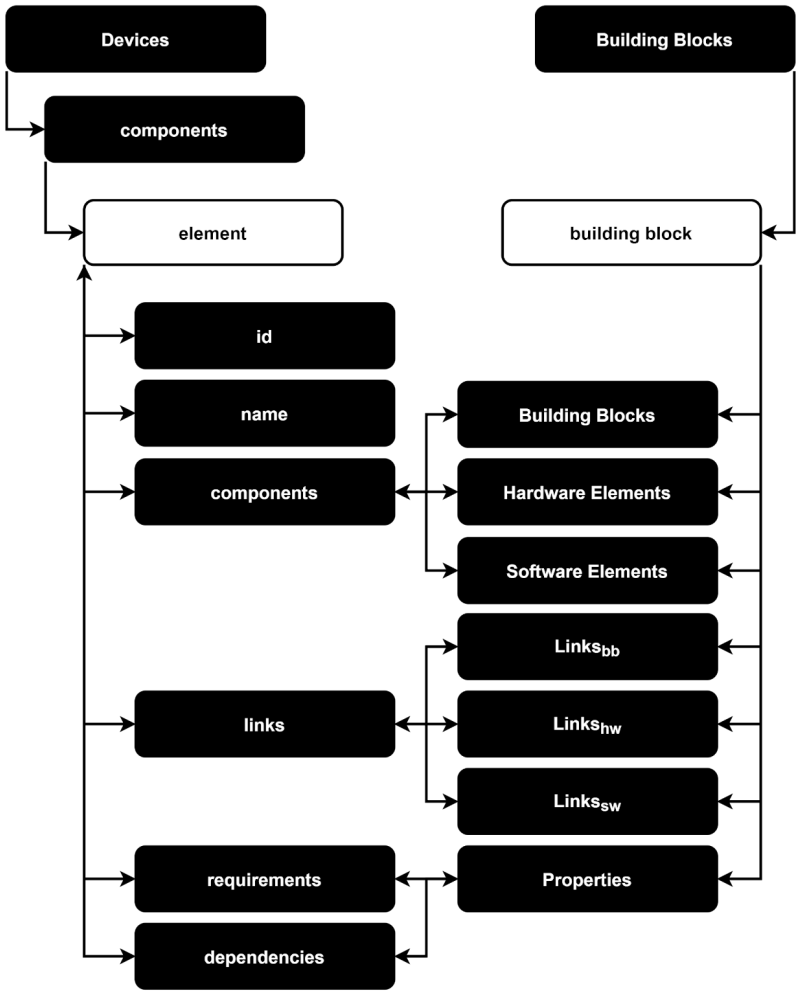

Figure 29. The mapping on the level of hardware and software elements

This algorithm can be used to generate an abstract system representation based on information about its devices, their elements and sub-elements. The output data is well-structured, while the algorithm takes into account the hierarchy of elements and dependencies between them as well as generates requirements for them.

It is important to note that this algorithm can be useful to an expert in the design of secure systems, but its full potential is revealed only when interacting with other algorithms from this chapter within the framework of the design methodology from Section 4.5. In this methodology other algorithms are providing input data, while the abstract model — output of this algorithm — is detailed by replacing abstract elements with their concrete implementations (taking into account requirements, mutual dependencies and possible conflicts indicated in the abstract model) based on the algorithm, presented in the following section.

4.4. Algorithm for the design of the detailed model of the system

The algorithm for the design of detailed models of microcontroller-based physical security systems is used to construct a detailed representation of a secure system based on its abstract representation. Detailed model of the system preserves and expands the structure of the abstract model of the system and takes into account compatibility, requirements, dependencies and hierarchy of system elements. The process of transition from the abstract system model to detailed one is a step-by-step process. Each step represents the process of selection of the concrete implementation of one of the system elements, while the sequence of steps is formed in accordance with hierarchy and dependencies between those elements. Moreover, after each step the number of options for further steps is limited in accordance with compatibility.

As **input data**, the algorithm takes the *abstract system model*. The structure of the abstract model of the system is JSON-based and contains the following fields:

- *devices*: data about each device of the system, including its unique key, id, name, components and recommendations;
- *recommendations*: data about recommendations to the implementation of the system to ensure its security against attackers with certain parameters, including unique key, id and name (description);
- *links*: data about links between devices of the system, including its unique key, id, type, parties, dependencies and requirements.

Each element from the “components” field has its unique key and id as well as data about its own components (sub-elements), links, requirements and dependencies. The abstract system model is provided by the previous algorithm, see Section 4.3.

As **output data**, the algorithm provides the *detailed system model*. The structure of the detailed model of the system is also JSON-based. Moreover, it has the same structure as the abstract model of the system but with some additions:

- each *element* from the *components* field that was selected is extended with the *selected* field: data about selected elements, including id, name and parameters of its implementation; parameters of the element differ for different components and controllers;
- each *device* of the system is extended with the *parameters* field: data about parameters of the designed device, including price, energy consumption, voltage, current, length, width, height, free memory and battery life; device parameters are based on parameters of its elements; parameters are mostly the same for all devices, however, the units for free memory are different for single-board computers and microcontrollers;
- each *link* between devices of the system is extended with the *selected* field: data about the selected links between devices, including id, name, interface, protocol and parameters; parameters are the same for each link and can be

divided into boolean and numerical ones; boolean parameters are defining if the selected link is wireless, directed, transfers data, charge or signal, requires access point, has encryption or authentication; numerical parameters are defining range and speed of the link.

The work process of the algorithm is automated, involvement of the operator is possible at the stage of selection of the concrete implementations of elements among suitable options provided by the algorithm. Alternatively, the algorithm can select concrete implementations on its own. Its overview is presented in Figure 30.

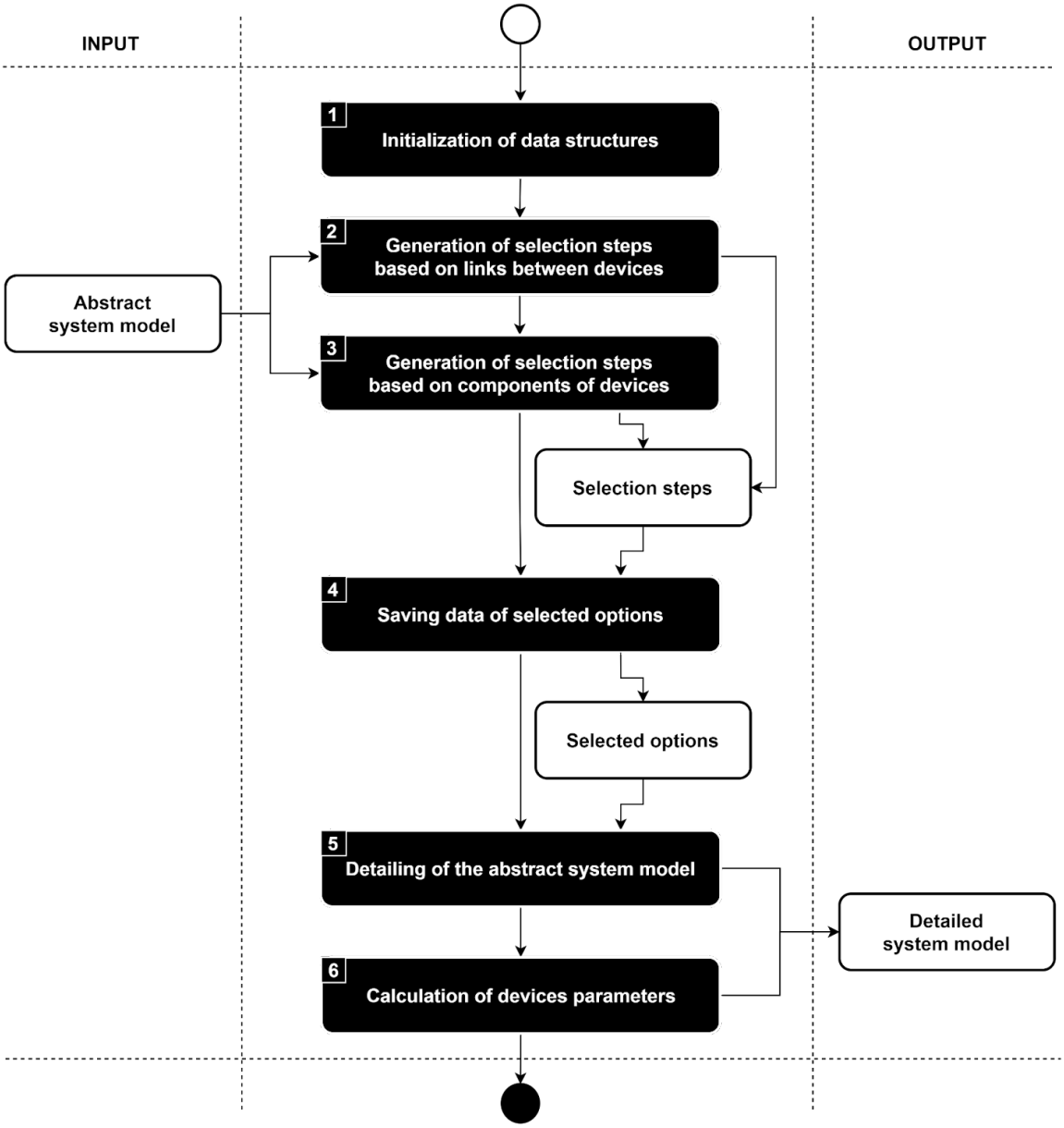

Figure 30. Overview of the algorithm for the design of detailed models

The work process of the algorithm contains **6 main stages**, namely, initialization of data structures, generation of selection steps based on links between devices, generation of selection steps based on components of devices, saving data of selected options, detailing of the abstract system model and calculation of the devices parameters. Let's consider them in more detail.

Stage 1: Initialization of data structures. This stage defines the data structures for storing the selection steps and selected options. There is no need to define the data structure for the detailed model of the system, because it is stored in the same data structure that was used for the abstract model of the system.

Data structure for selection steps is JSON-based and contains unique keys for each step of selection. Using this key, data about the selected element can be extracted. Each selected element has key, type, id, name, label, hierarchy, dependencies and requirements. There is also an additional field "selected" to store data about the selected options as well as the field "same for" that prevents the selection of one element multiple times.

Data structure for selected options is JSON-based and contains keys table and database id. By the table key it is possible to extract data about the database table, where data on the selected option is stored, while database id is identificator of the concrete data tuple in the database table.

Stage 2: Generation of selection steps based on links between devices. At this stage the sequence of selection steps is filled with data about selection of links between devices of the system. The sequence of selection steps is a very important part of the algorithm because of dependencies between components of devices as well as possibility of their conflicts in terms of compatibility. That is why the generation of selection steps starts with selection of links between devices. Each link, after its selection, is limiting options for controllers and components that are related to communications between devices for compatibility.

Stage 3: Generation of selection steps based on components of devices. This stage is aimed at filling the sequence of selection steps with data about components of devices. This process is more complicated because of the hierarchical nature of device components compositions in the abstract model. In addition, it is important to take into account that components of one device can depend on the selection of components of another device. That is why firstly devices are selected in some order too, while data about each device component composition is extracted recursively. Moreover, the sequence of extracted components is also based on their hierarchy. Each element, after its selection, is limiting options for its dependable elements. For example, selection of the controller is limiting options for components that are connected to it for compatibility.

Stage 4: Saving data of selected options. At this stage the process of selection of concrete implementations begins. Each selection step means the choice of one option among suggestions. This process can be manually done by the operator or automatically by the algorithm. After the option is selected, the choice is saved, so it would be taken into account during the selection of other elements that have dependencies with the selected one. For example, if the link responsible for communication between devices of the system is representing Wi-Fi connection, the options for controllers are limited to those ones that support Wi-Fi or can be extended to support it. List of options is based on the content of the database, while it can be limited according to requirements of the abstract representation of the selected element. For example, requirements for the controller can limit its options to those that have at least the necessary amount of flash memory and pins. So, during this stage all options that are representing the abstract element are limited in accordance with compatibility, requirements and dependencies.

Stage 5: Detailing of the abstract system model. This stage is aimed at filling the abstract system model with the data of selected implementations of its elements and represents the process of detailing. As we mentioned during output data description, each selected element is extended with the selected field. This extension is based on the content of the database, while selected options data structure provides data on the table where content is stored as well as id of its tuple. For example, element with name “single-board computer” can have selected field with the following key-values: Raspberry Pi 4 Model B 2GB, Broadcom BCM2711 1.5 GHz, Cortex A72 4-core 64-bit, 2GB RAM, 5V, 3A, 85x56x17 mm, 69 euro, 540 mA. The situation for each selected link is the same. For example, the link related to Wi-Fi connection between devices can have selected fields with the following key-values: Wi-Fi IEEE 800.11 2.4GHz WPA2-PSK, 40 meters range, 20 Mbit/s.

Stage 6: Calculation of the devices parameters. At this stage the parameters of devices of the designed system are calculated. As was mentioned in the output data description, those calculations are based on the parameters of the elements of devices and are mostly the same for all devices. For example, parameters of the device that is representing a server of the system can be as follows: 106 euro, 540 mAh, 5V, 3A, 85x153.5x44.5 mm, 29400 MB of free memory, 37 hours of battery life. Note that parameters of the system as a whole are not calculated, because the necessary amount of its devices is not known by the algorithm and depends on the concrete implementation of the designed system.

It is important to note that the algorithm relies on the content of the database when extracting options that can be selected as well as when checking parameters of the selected links and elements. It means that the correctness of its work strongly depends on the content of the database. The possible structure of such a database is presented in detail in Chapter 5.

The detailed system model is a mapping of the extendable set-based hierarchical relational model of microcontroller-based physical security systems from Chapter 3. Moreover, the detailed system model is an extension of the abstract system model. That is why in this section only differences between detailed and abstract models are shown, while mapping is shown in the previous one.

As we mentioned during the output data description, the differences are on the level devices, their elements and links between them. Changings in the mapping on the level of links are presented in Figure 31, while on the level of devices — in Figure 32 and elements — in Figure 33.

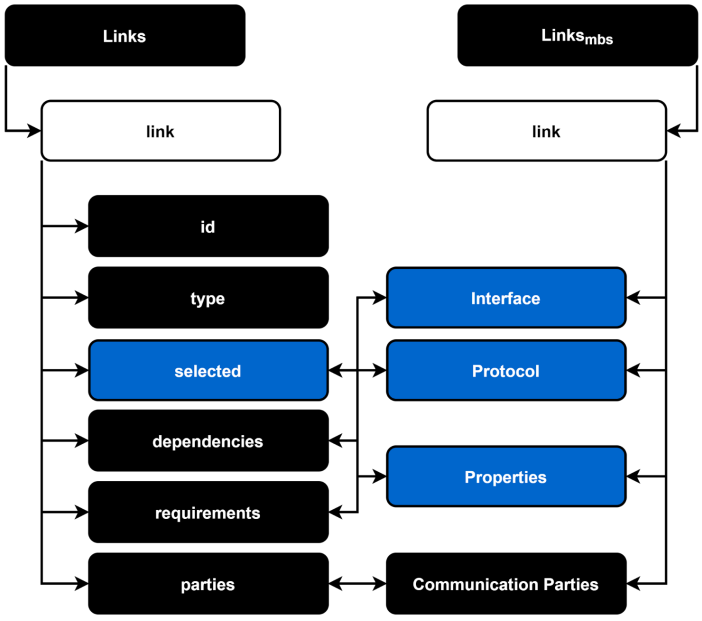

Figure 31. Changings in the mapping on the level of the links between devices

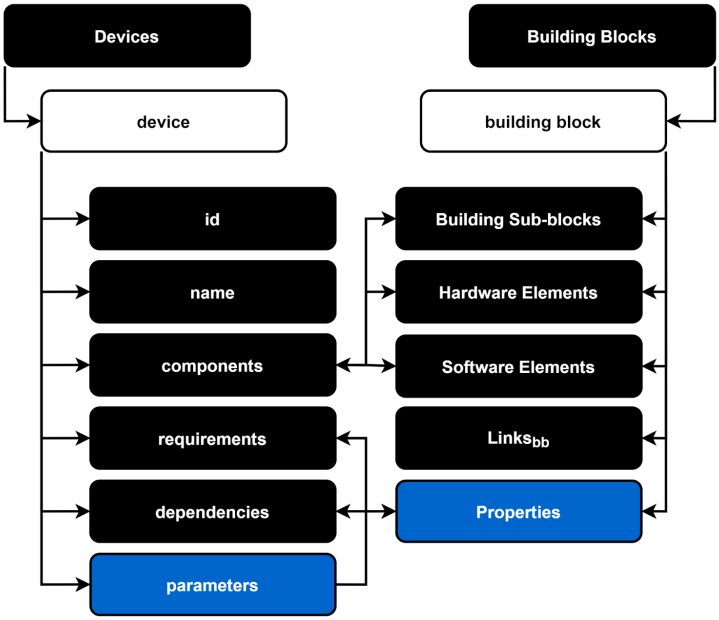

Figure 32. Changings in the mapping on the level of devices

Note that links between building blocks that are representing devices are taken into account in the separate part of the detailed system model.

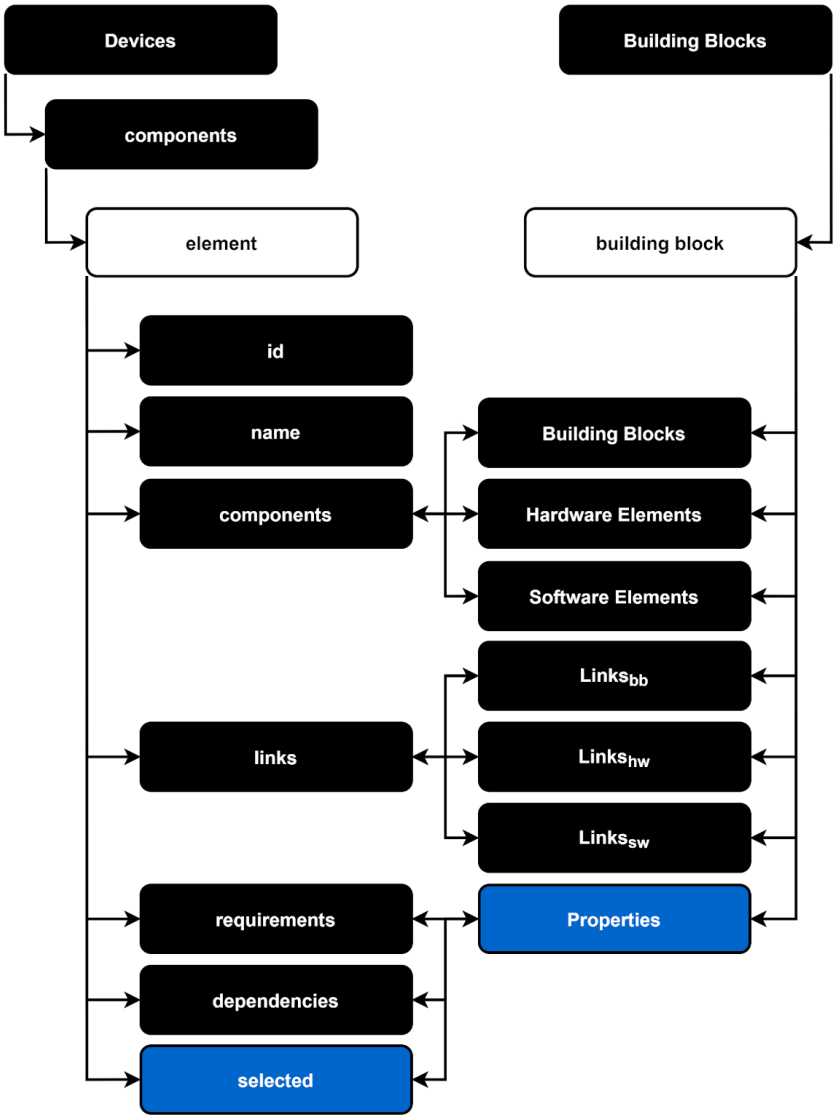

Figure 33. Changings in the mapping on the level of elements

This algorithm can be used to generate the detailed system representation based on the abstract one. The output data is well-structured, while the algorithm takes into account compatibility, requirements, dependencies and hierarchy of elements.

It is important to note that this algorithm can be useful to an expert in the design of secure systems, but its full potential is revealed only when interacting with other algorithms from this chapter within the framework of the design methodology from Section 4.5. In this methodology, this algorithm is using the input data — abstract system model — that is provided based on work of other algorithms and details it with the selection of concrete implementations of components and controllers. Moreover, the algorithm calculates the parameters of the system devices.

4.5. Methodology for the design of the system

The methodology for the design of microcontroller-based physical security systems consists of two main cycles. The main goal of the first cycle is to design the abstract system model based on provided requirements, while the second one is about the design of the detailed system model based on selection of components. The key idea of the methodology is in providing reasonable secure solutions. Such solutions are called alternatives and built according to functional requirements and non-functional limitations. These requirements and limitations are obtained through transformation of the stakeholders wishes inside of the requirements and limitations formation technique. The developed methodology considers components that are improving the security level as an integral part of the system. Moreover, the suggested solution works with non-security parameters of the system according to the blackbox principle: the methodology needs to know how many resources the system requires to perform its functions, so it would be able to calculate the number of resources available for components that are improving the system security level.

Each cycle of the methodology consists of the testing process and seven stages that are associated with the developed extendable set-based hierarchical relational model from Chapter 3. The testing process occurs after each stage as many times as necessary to build the model of the system.

The objective of the testing process is in checking constructed models in terms of their correctness and compatibility. In terms of the input data, the first cycle works with requirements and limitations, while providing abstract models of system elements and the abstract model of the system as an output. In its turn, the second cycle works with models that were designed by the first cycle and adds to the abstract model data about selected devices and their parameters as an output. It is also possible that the first cycle in addition to requirements and limitations will take the model of the system as an input data if the goal is to improve its security according to the new wishes of the stakeholder.

The idea of the design of the abstract system model cycle is to find out an abstract composition of the system to fulfill all formed requirements and limitations. In its turn, the fulfillment of all formed requirements and limitations would mean that the designed system has all necessary abilities to be able to solve all general tasks. And if the designed system is able to solve all general tasks, then this is exactly the system that the stakeholder wanted, assuming that the decisions are made correctly. That is why each design stage begins from the analysis of the provided input.

To simplify the understanding of the abstract system model design cycle, the input and output data for each stage were summarized in Table 9, while its overview is presented in Figure 34.

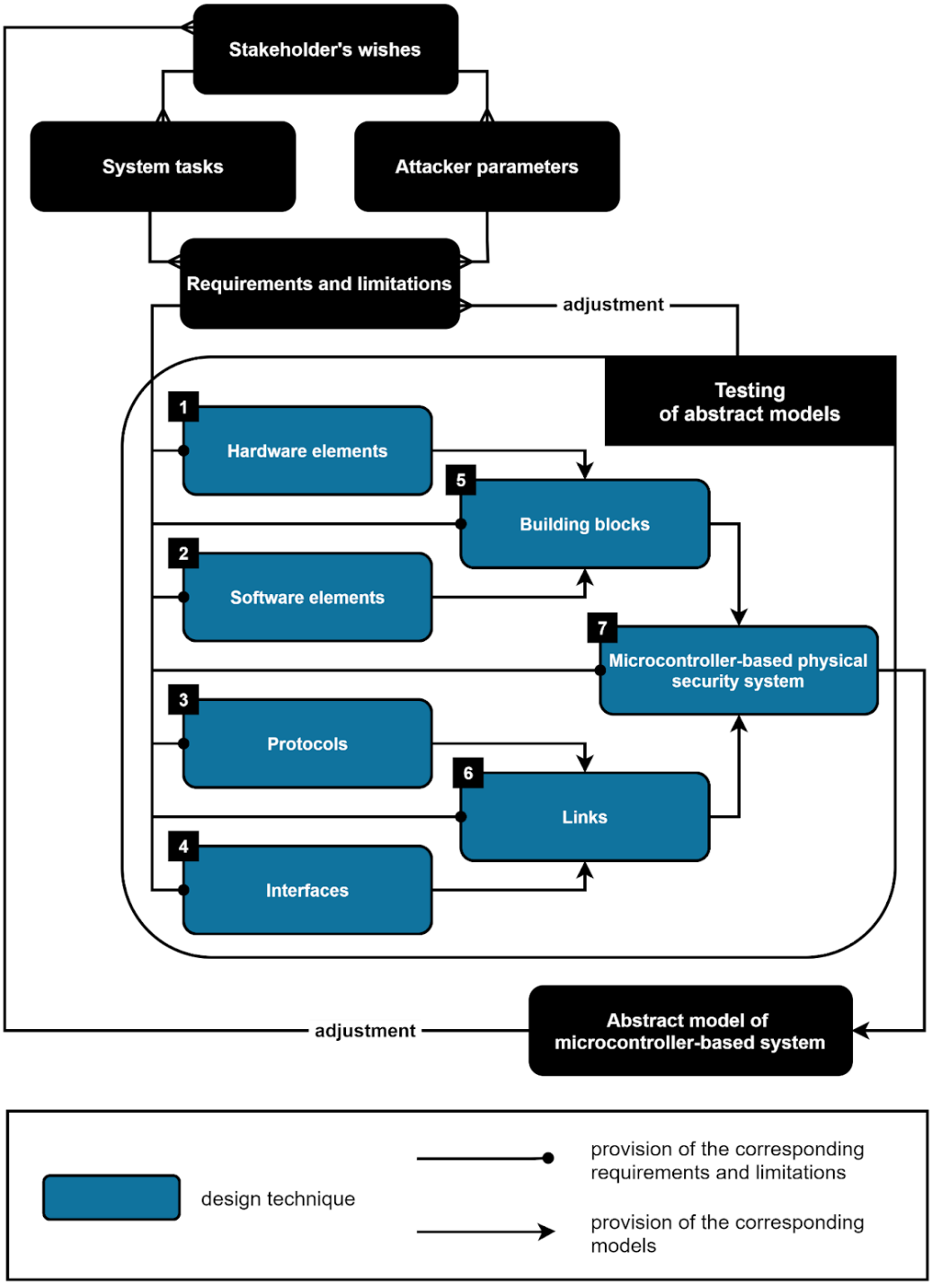

Figure 34. Overview of the abstract system model design cycle

Understanding of the dependencies between tasks, abilities, requirements, limitations and different abstract elements of their fulfillment helps the methodology to build the system design process step-by-step. In addition, it allows one to reduce the number of returns to one of the previous steps for reviewing the decisions made.

The issue is that mostly such dependencies are not linear. For example, different hardware elements can be used to fulfill several requirements at the same time, while software elements may have their own requirements. It means that there are a number of possible interconnections as well as conflicts between system elements that should be taken into account.

Table 9. Input and output data of the abstract system model design cycle

	design technique	input	output
1	hardware element	requirements and limitations, hardware sub-elements models (null is possible)	model of hardware element
2	software element	requirements and limitations, software sub-elements models (null is possible)	model of software element
3	protocol	requirements and limitations	model of protocol
4	interface	requirements and limitations	model of interface
5	building block	requirements and limitations, hardware sub-elements models (null is possible), software sub-elements models (null is possible), building sub-blocks models (null is possible)	model of building block
6	link	requirements and limitations, model of protocol, model of interface	model of link
7	system	requirements and limitations, building blocks, links models, sub-systems models (null is possible)	model of system

As soon as the methodology constructed the scenario of building an abstract model of the microcontroller-based physical security system according to its elements hierarchy and nesting, each design technique starts to work. And while the first four of them are working only with requirements and limitations, the following ones are taking models from the previous stages as an input too. And because the suggested solution has a strong focus on security, the list of possible harmful effects on the system and its elements is also analyzed during each stage.

It is important to note that the methodology works with models of attacker and attack actions to take security into account, see Chapter 3. The presence of such models allows the methodology to transform stakeholder's wishes for security into the requirement like "system should be secure against the attacker with certain parameters", while the list of possible attack actions for such an attacker could be obtained from the attack surface that is also a part of the database. And according to the requirements the methodology has a possibility to reduce the number of attack actions that are taken into account during the system security design process.

The testing process for abstract models of the system and its elements occurs after each use of one of the corresponding design techniques to check the correctness and compatibility of the designed model. It is important to note that according to the result of the analysis, it might be concluded that the designed element does not fit on the system level — some of the requirements or limitations are violated. It might happen because of elements incompatibility (platforms, architectures, interfaces, voltage etc.) or due to the lack of computing power of other system devices. In such a situation, the methodology would suggest partial changes in the requirements and limitations or on refusal from some of them. Reconsideration of the requirements and limitations means the need for a redesign process that can affect not only the current design phase, but also previous ones. This process occurs as many times as necessary to build the abstract model of the microcontroller-based physical security system or to confirm the inability to do so.

If the abstract model of the microcontroller-based physical security system is constructed, the output of the first cycle is taken as input by the detailed system model design cycle. To simplify its understanding, the input and output data for each stage were summarized in Table 10, while its overview is presented in Figure 35.

The idea of the detailed system model design cycle is to fill the abstract model of the system with concrete implementations of components and controllers, their firmware and software. The selection process is based on functional requirements and non-functional limitations as well as dependencies between system elements that were formed during the abstract system model design cycle. Moreover, each selected component forms additional restrictions on the choice of the subsequent. The fundamental difference between this cycle and the previous one is that the components have specific parameters — price, energy efficiency, size, computing power, number of pins, flash memory, etc.

Each stage of the detailed model design cycle begins with the analysis of the provided abstract model. Dependencies between system elements, their requirements, compatibility, hierarchy and nesting were taken into account during the abstract system model design cycle and therefore are an internal part of the model. With availability of such information and data about possible options of elements from the database the methodology is able to build the system components selection process step-by-step. An important difference of this process is the choice of elements from those defining the system as a whole (for example, data transfer environment between devices of the system) to the elements that are implementing the separate functionality of the devices (for example, motion sensor).

Note that depending on the chosen option, some selection stages might be skipped based on the number of available alternatives of the component. Moreover, in most situations the methodology would use already existing implementations of controllers and components instead of construction of the new integrated circuits.

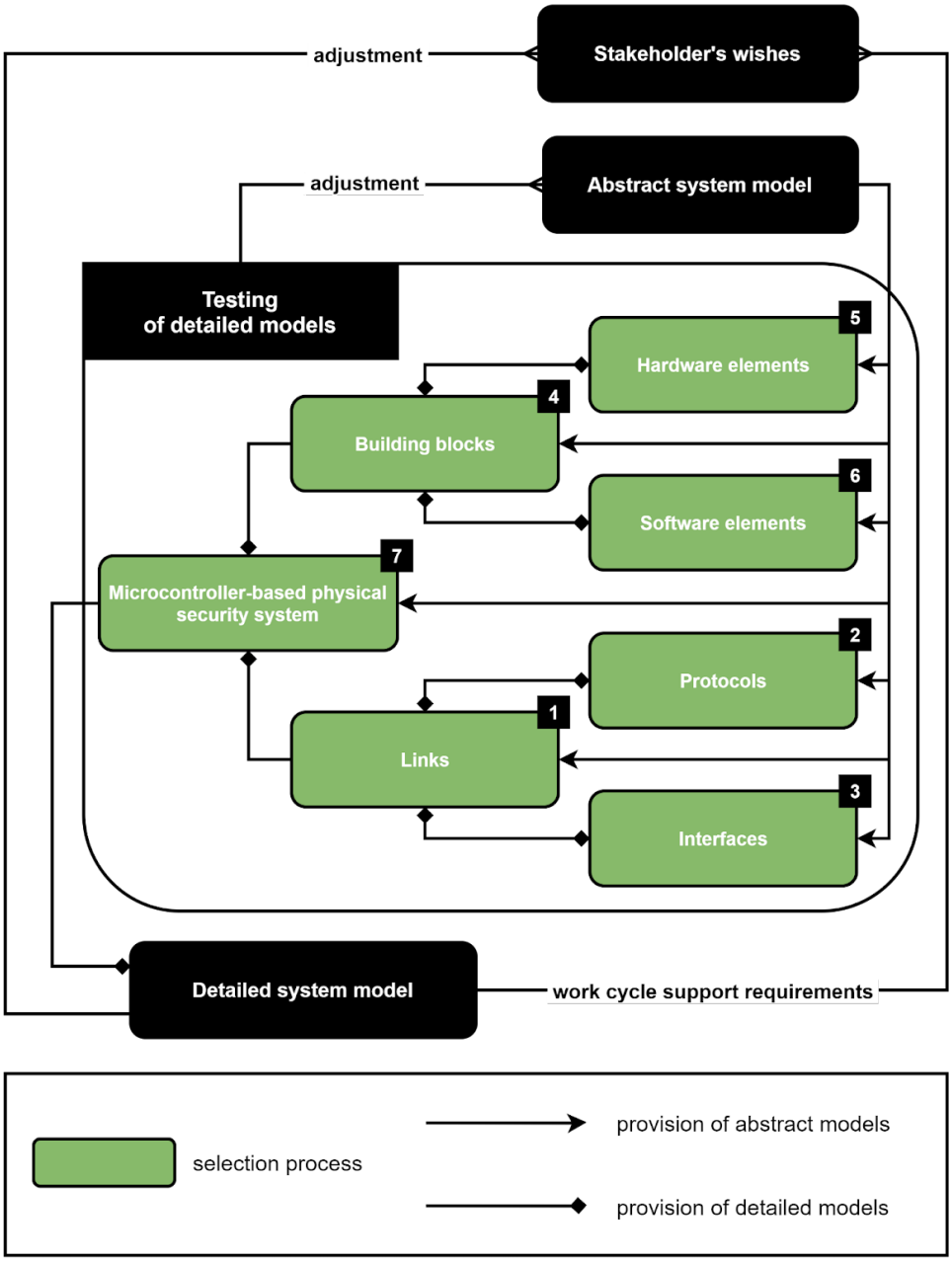

Figure 35. Overview of the detailed system model design cycle

On the other hand, there is still a possibility when the extension of the functionality of the already available solutions would be required. Such a situation would most likely happen on the level of device-to-device communication. This is due to the fact that protocols used in such a communication are generally determined only by interface, packet size and address range [133]. Therefore, it becomes necessary to implement additional functionality on the top of these protocols including, for example, dynamic addressing, packet size extension, secure sessions, encryption of the transmitted data as well as mutual authentication between devices. And if the database does not contain elements that satisfy such functionality, the methodology translates them into requirements for the implementation stage of the system.

Table 10. Input and output data of the detailed system model design cycle

	selection process	input	output
1	link	link model, already selected elements of the system (null is possible)	selected link with parameters
2	protocol	protocol model, already selected elements of the system (null is possible)	selected protocol with parameters
3	interface	interface model, already selected elements of the system (null is possible)	selected interface with parameters
4	building block	building block model, already selected elements of the system (null is possible)	selected building block with parameters
5	software element	software element model, already selected elements of the system (null is possible)	selected software element with parameters
6	hardware element	hardware element model, already selected elements of the system (null is possible)	selected hardware element with parameters
7	system	system model, already selected elements of the system (null is possible)	system model with parameters

The testing process for selected components of the detailed model occurs after each use of one of the corresponding selection techniques to check its correctness and compatibility. This process checks the properties of the system that are only apparent when concrete components, controllers, interfaces and protocols are selected. It is achievable through the monitoring of the already selected components to find out, for example, incompatibility or lack of memory size, computing power, battery capacity, etc.

It is important to note that according to the result of the analysis, it might be concluded that the selected components are not working as intended — some of the requirements or limitations are violated. It might happen due to the lack of information about changes in the element properties during its operation under specific conditions. In such a situation, the methodology will have to rebuild the system model until it is done.

The work process of the methodology is mostly automated, involvement of the operator is required during transformation of wishes of stakeholder into requirements and limitations and optional at the stage of selection of the concrete implementations of elements among suitable ones during the process of detailing the abstract system model. Alternatively, the methodology can select implementations on its own.

Another way to represent the workflow of the methodology, is to showcase its connection with algorithms, described in previous sections of this chapter:

1. formation of requirements for the system, see Section 4.1;
2. formation of the system component composition, see Section 4.2;
3. design of the abstract model of the system, see Section 4.3;
4. design of the detailed model of the system, see Section 4.4.

First three algorithms are representing the abstract system model design cycle, while the last one is representing the detailed system model design cycle, see Figure 36.

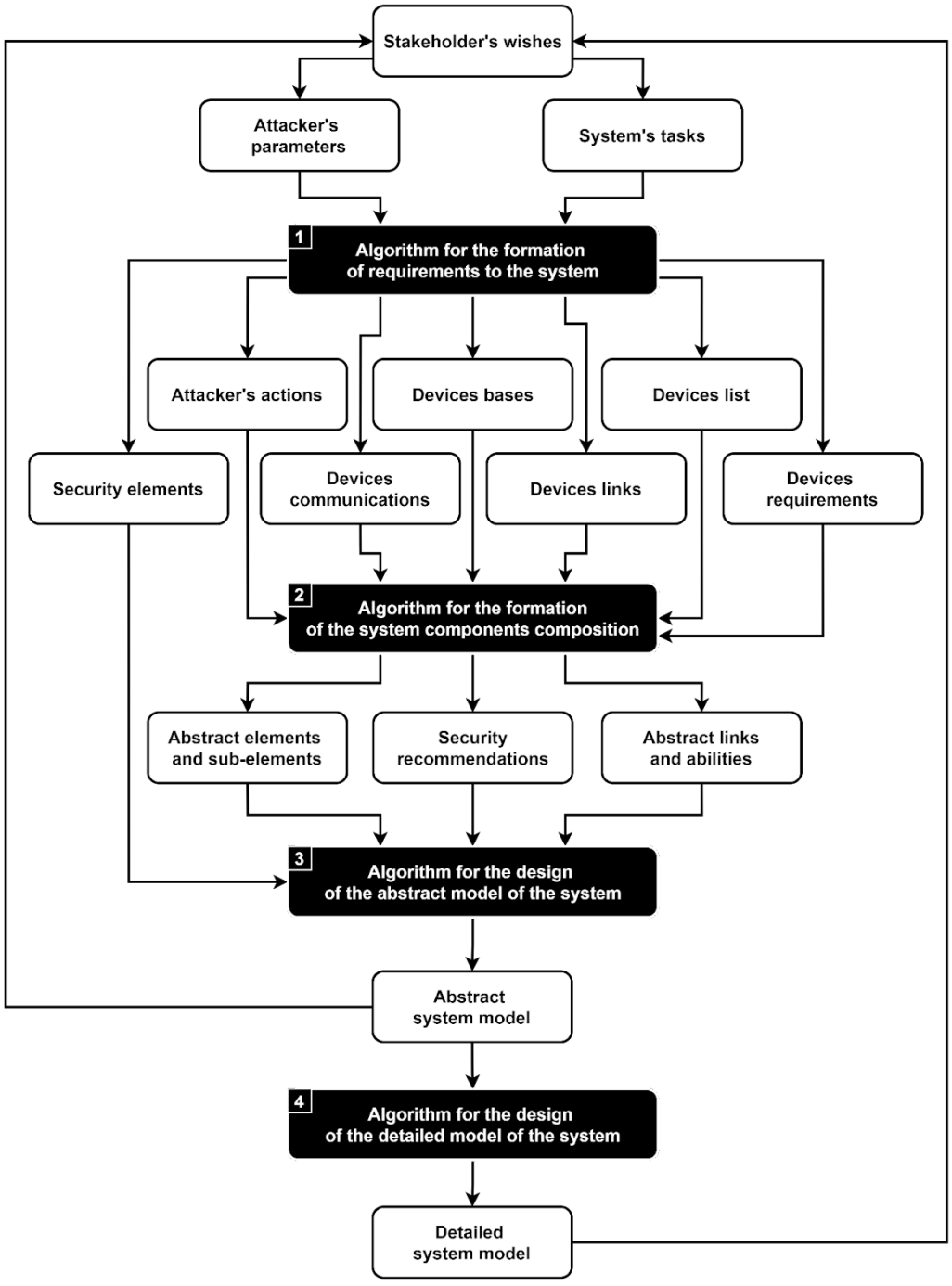

Figure 36. Connections between the methodology and developed algorithms

The main idea of the developed methodology is to provide an automated tool for the design of microcontroller-based physical security systems that are protected against attackers. This methodology allows one to reduce the number of weak places and architectural defects, thereby significantly reducing the attack surface of the microcontroller-based physical security systems. In turn, this will reduce the security risks that can lead to financial losses, loss of time as well as the safety of people.

It is important to note that the methodology is not aimed to replace security experts. In most situations, an expert in the security of microcontroller-based systems knows about existing best and highly specialized solutions and is able to form alternatives at a very high level, while the quality of the solution provided by the methodology directly depends on the correctness and completeness of the database. But it can be useful for an expert to automate routine tasks and provide alternative solutions.

4.6. Conclusions on Chapter 4

The algorithm for the formation of requirements for microcontroller-based physical security systems is used to extract attack actions that are possible for the attacker and a list of devices of the designed system, their links, communications, bases and requirements in accordance with the attacker's parameters and system’s general tasks. This algorithm works with abstract requirements that can represent components of devices and their sub-components as well as links between devices, taking into account controllers used as the basis of the device and possible for each device types of communications that determine attack actions that are potentially dangerous for the designed devices. The output data is well-structured and JSON-based. The work process of the algorithm is automatic, the operator is required for the translation of wishes of the stakeholder into the attacker’s parameters and general tasks of the system. The work process of the algorithm contains 6 main stages, namely, initialization of data structures as well as getting attack actions possible for the attacker, security elements to prevent attack actions, abilities of the designed system, requirements of the designed system and device data. Last stage is divided into 7 sub-stages, namely, getting device name, tasks, abilities, requirements, base, types of communication and links.

The novelty of the algorithm for the formation of requirements for the system is in retrieving a list of microcontroller-based system devices, communications available to them, as well as requirements for them only on the basis of system tasks, while the list of attack actions that are possible for the attacker is retrieved in accordance with the type of access, knowledge and resources the attacker has.

The algorithm for the formation of the microcontroller-based physical security system component composition is used to extract abstract elements and sub-elements of the devices of the system, security recommendations to the system and its devices implementation as well as abstract links between devices with related to them

abilities based on attack actions that are possible for the attacker, list of devices of the system, their bases, types of communications and links, requirements for them. This algorithm works with abstract elements, links and recommendations and represents the designed system components compositions as multiple devices, each of which has multiple abstract elements, while each abstract element can have multiple abstract sub-elements. Wherein abstract elements and sub-elements are representing controllers and components as well their software, including those that are related to security. The output data is well-structured and JSON-based. The work process of the algorithm is automated, the operator is not required. The work process of the algorithm contains 2 main stages, namely, initialization of data structures as well as getting component composition of devices. Last stage is divided into 5 sub-stages, namely, getting abstract elements with their sub-elements, possible attack actions, additional abstract elements with their sub-elements, security recommendations to implementation as well as links between devices.

Unlike other solutions, the algorithm for the formation of the system component composition is retrieving abstract elements and sub-elements of the designed microcontroller-based system in accordance with the requirements, device base and already retrieved elements, while security elements are represented as abstract elements, sub-elements, and recommendations for the system implementation.

The algorithm for the design of abstract models of microcontroller-based physical security systems is used to construct an abstract representation of a secure system based on its devices list, their abilities, elements and sub-elements as well as security recommendations. This algorithm represents the system as an abstract hierarchical model that takes into account connections between system devices, their elemental composition, dependencies between device elements and requirements for them. As output data, the algorithm provides the abstract system model that contains abstract system representation. The structure of the abstract model of the system is JSON-based and contains the following fields: devices, recommendations and links, while each element of the device from the “components” field has its own components (sub-elements), links, requirements and dependencies. The work process of the algorithm is automatic, the operator is not required. It contains 7 main stages, namely, abstract model initialization as well as generation of system recommendations, devices, links requirements, dependencies and hierarchy.

The novelty of the algorithm for the design of the abstract model of the system is in taking into account complex dependencies between the elements of microcontroller-based systems, namely, their hierarchy, nesting, communications, conflicts and requirements. Moreover, this algorithm is not limited to specific platforms and architectures and because of its abstract nature reduces the number of parameters to be searched, thereby increasing the work speed of the solution.

The algorithm for the design of detailed models of microcontroller-based physical security systems is used to construct a detailed representation of a secure system based on its abstract representation. Detailed model of the system preserves and expands the structure of the abstract model of the system and takes into account compatibility, requirements, dependencies and hierarchy of system elements. The process of transition from the abstract system model to detailed one is a step-by-step process. Each step represents the process of selection of the concrete implementation of one of the system elements, while the sequence of steps is formed in accordance with hierarchy and dependencies between those elements. Moreover, after each step the number of options for further steps is limited in accordance with compatibility. As output data, the algorithm provides the detailed system model. The structure of the detailed model of the system is also JSON-based. Moreover, it has the same structure as the abstract model of the system but with some additions: each element from the components field that was selected is extended with the selected field; each device of the system is extended with the parameters field; each link between devices of the system is extended with the selected field. The work process of the algorithm is mostly automated, involvement of the operator is possible at the stage of selection of the concrete implementations of elements among suitable options provided by the algorithm. Alternatively, the algorithm can select concrete implementations on its own. The work process of the algorithm contains 6 main stages, namely, initialization of data structures, generation of selection steps based on links between devices, generation of selection steps based on components of devices, saving data of selected options, detailing of the abstract system model and calculation of the devices parameters.

Unlike existing solutions, the algorithm for the design of the detailed model of the system makes it possible to form a step-by-step process of detailing the abstract representation of microcontroller-based physical security systems in accordance with the hierarchy and mutual dependencies of their elements. Moreover, this algorithm calculates the parameters of the system devices based on the parameters of their elements as well as the parameters of the system based on the parameters of its devices. This algorithm does not replace the abstract model of the system, but expands and complements it.

The methodology for the design of microcontroller-based physical security systems consists of two main cycles. The main goal of the first cycle is to design the abstract system model based on provided requirements, while the second one is about the design of the detailed system model based on selection of components. Each cycle of the methodology consists of the testing process and seven stages that are associated with the developed extendable set-based hierarchical relational model from Chapter 3. The testing process occurs after each stage as many times as necessary to build the model of the system. The objective of the testing process is in checking constructed models in terms of their correctness and compatibility. In terms of the input data, the first cycle works with requirements and limitations, while

providing abstract models of system elements and the abstract model of the system as an output. In its turn, the second cycle works with models that were designed by the first cycle and adds to the abstract model data about selected devices and their parameters as an output.

Another way to represent the workflow of the methodology is to showcase its connection with algorithms, described in this chapter, namely, formation of requirements for the system, formation of the system components composition, design of the abstract model of the system and design of the detailed model of the system. The first three algorithms are representing the abstract system model design cycle, while the last one is representing the detailed system model design cycle.

The novelty of the methodology for the design of microcontroller-based physical security systems lies in a new approach to the design, which allows combining various design techniques on the basis of hierarchical relational model transformation algorithms. Moreover, the suggested approach is modular and extensible, takes into account the security of the physical layer of the system, works with the abstract system representation and is looking for a trade-off between the security of the final solution and expended resources. Also, unlike existing solutions, the methodology has a strong focus on security. It is aimed at ensuring the protection of the system against attacks at the design stage, considers security components as an integral part of the system and checks if the system can be designed in accordance with given requirements and limitations.

One of the main findings of this work, namely software implementation of the methodology for the design of microcontroller-based physical security systems is presented in the next chapter. This software is used to validate the correctness of the developed methodology.

Chapter 5. Software implementation of the methodology for the design of microcontroller-based physical security systems

This chapter describes the software implementation of the methodology for the design of microcontroller-based physical security systems. The description contains information about the architecture, database, script and interface of the application. The application was developed to validate the correctness of the methodology.

5.1. Architecture of the software implementation

Software implementation of the methodology is an application that consists of Python script [135], PostgreSQL database [136] and Tkinter [138] interface. The overview of the software implementation architecture is presented in Figure 37.

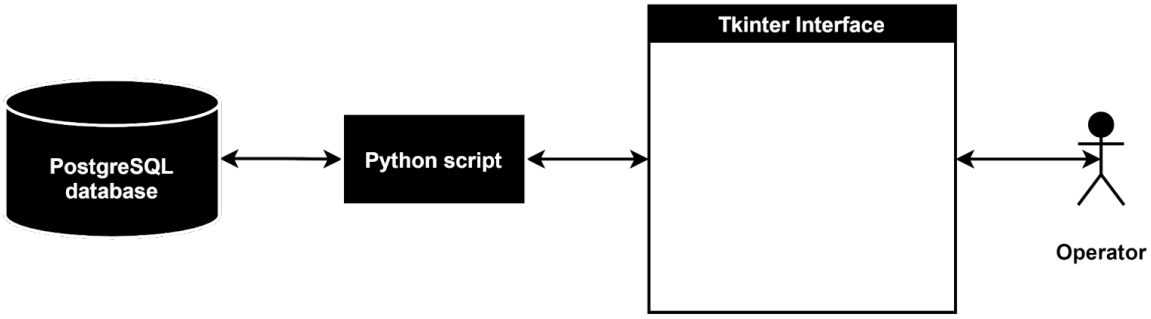

Figure 37. The architecture of the software implementation

PostgreSQL database is required to store data about the extendable set-based hierarchical relational model of microcontroller-based physical security systems from Chapter 3, as well as data for algorithms and methodology from Chapter 4. This data helps to provide data to the operator as well as helps algorithms and methodology to make decisions about elements compatibility, dependencies, hierarchy and nesting. For more detail, see Section 5.2.

Python script represents the implementation of the algorithms and methodology from Chapter 4. Each algorithm is implemented as a number of functions, while all functions are connected with each other in a single methodology. The connection between the Python script and the PostgreSQL database is provided by the library. For more detail, see Section 5.3.

Tkinter interface is required to receive input data from the operator, namely, parameters of the attacker and tasks of the designed system, as well as to provide the output data to the operator. For more detail, see Section 5.4.

5.2. Database of the software implementation

The developed database contains more than 90 tables, while the database initialization contains more than 2300 lines of PL/pgSQL queries [144].

The developed database has too many tables to be shown at once, that is why it was decided to divide its description on the following parts:

1. Storage of the attacker, attack actions and security elements, see Chapter 3.
2. Storage of tasks, abilities and requirements, see Chapter 4.
3. Storage of abstract elements, sub-elements and links, see Chapter 4.
4. Storage of detailed elements, see Chapter 4.

Let's consider each part of the database in more detail.

5.2.1. Storage of the attacker, attack actions and security elements

The structure of the database for this part contains 11 tables, see Figure 38.

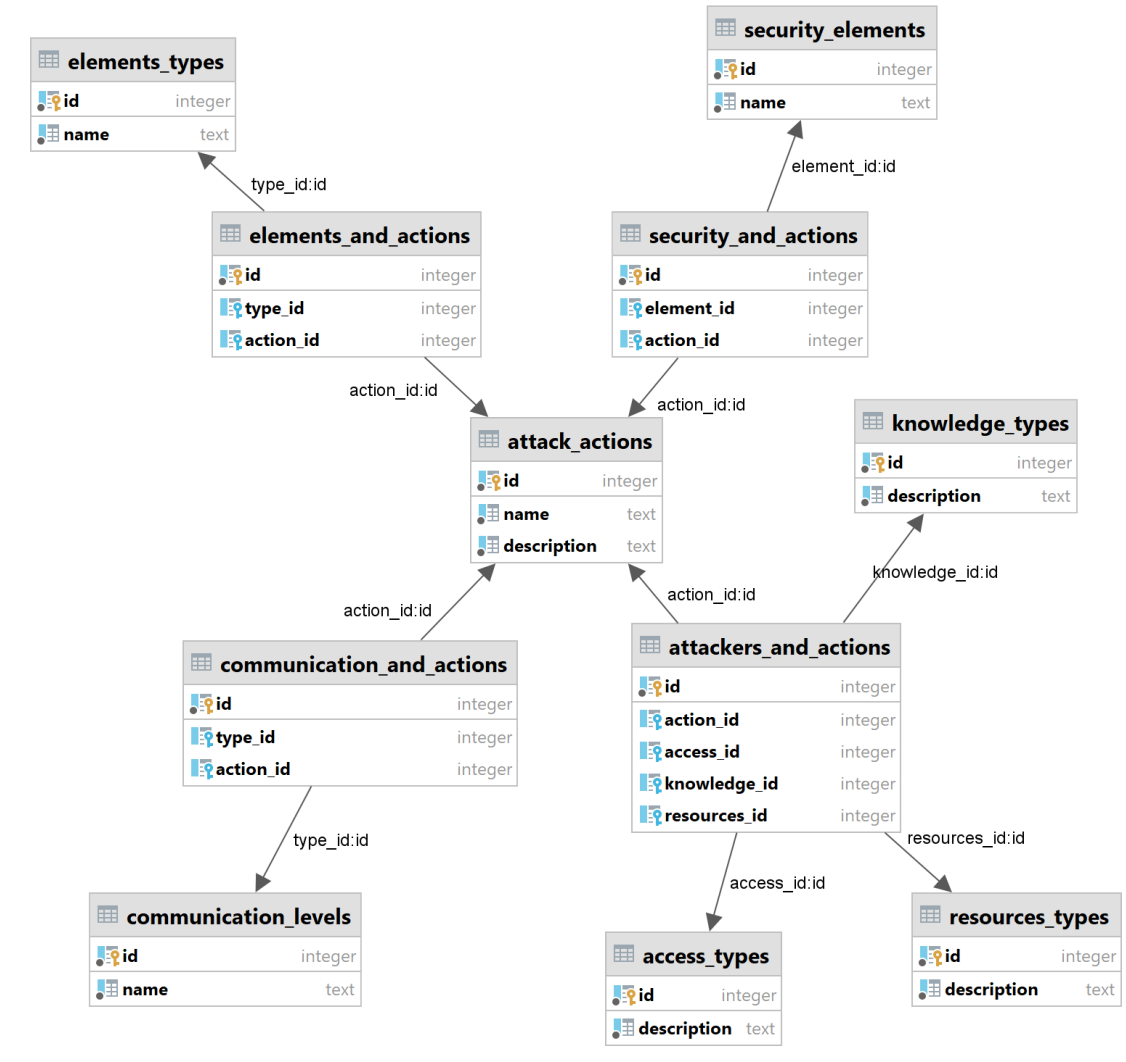

Figure 38. Database structure: attacker, attack actions and security elements

Parameters of the attacker are stored in *access_types*, *knowledge_types* and *resources_types* tables of the database. Each table represents corresponding parameters with the help of a unique identificator and description. The content of each table is based on the attacker model provided in Chapter 3, see Figure 39.

Table: access_types

	id	description
1	1	No access to the system
2	2	Access to the system through global networks
3	3	Access to the system, its devices and interfaces through local networks
4	4	Physical access to the system, its devices and interfaces
5	5	Full access to the system and its elements

Table: knowledge_types

	id	description
1	1	Knowledge about the system from publicly available sources
2	2	Knowledge about system parameters
3	3	Knowledge about means of protection of the system
4	4	Knowledge about system software and hardware

Table: resources_types

	id	description
1	1	An attacker can use only widely-spread software tools and exploit only known vulnerabilities
2	2	An attacker can use specialized software tools and exploit previously non-used vulnerabilities
3	3	An attacker has a possibility to investigate the system

Figure 39. Content of the database: access, knowledge and resources types

Attack actions are stored in the *attack_actions* table of the database. This table represents attack actions with the help of a unique identificator, name and description. The content of the table is based on the attack actions model provided in Chapter 3, see Figure 40.

	id	name	description
1	1	gie	generation of incorrect component events
2	2	bcd	bypassing component detection algorithms
3	3	rpt	replacement of the component
4	4	rmt	removement of the component
5	5	rfw	replacement of the firmware
6	6	rbl	reinstallation of the bootloader
7	7	mup	malfunction of the update system
8	8	imw	interception, modification or termination of wired communications
9	9	vau	violation of the authentication system
10	10	cad	cryptographic analysis of transmitted data
11	11	iec	increased energy consumption
12	12	iws	interception, modification or termination of wireless communications
13	13	soc	social engineering
14	14	pwr	power failure
15	15	web	disruption of web services
16	16	dbd	database disruption

Figure 40. Content of the database: attack actions

The connections between parameters of the attacker and possibility to implement attack actions are stored in the *attacker_and_actions* table of the database. The content of the table is unique combinations of ids from *access_types*, *knowledge_types*, *resources_types* and *attack_actions*, see Figure 41.

	id	action_id	access_id	knowledge_id	resources_id
1	1	1	3	2	3
2	2	2	3	3	2
3	3	3	4	2	1
4	4	4	4	1	1
5	5	5	4	4	3
6	6	6	4	4	3
7	7	7	3	3	2
8	8	8	4	1	1
9	9	9	3	3	2
10	10	10	3	3	2
11	11	11	3	2	1
12	12	12	3	1	2
13	13	13	1	1	1
14	14	14	4	1	1
15	15	15	2	2	1
16	16	16	2	2	1

Figure 41. Content of the database: attackers and actions

Based on such a table, it is possible to extract attack actions that are possible for the attacker in accordance with his or her parameters (for example, access type — 3, knowledge type — 3, resources type — 3), with the help of the following sequence:

```
SELECT attack_actions.*
FROM attack_actions, attackers_and_actions
WHERE
 attackers_and_actions.action_id = attack_actions.id
AND attackers_and_actions.access_id <= 3
AND attackers_and_actions.knowledge_id <= 3
AND attackers_and_actions.resources_id <= 3
ORDER BY attack_actions.id;
```

The output of such an SQL sequence is presented in Figure 42.

	id	name	description
1	1	gie	generation of incorrect component events
2	2	bcd	bypassing component detection algorithms
3	7	mup	malfunction of the update system
4	9	vau	violation of the authentication system
5	10	cad	cryptographic analysis of transmitted data
6	11	iec	increased energy consumption
7	12	iws	interception, modification or termination of wireless communications
8	13	soc	social engineering
9	15	web	disruption of web services
10	16	dbd	database disruption

Figure 42. SQL sequence: attack actions that are possible for the attacker

The possibility of the implementation of attack actions depends not only on parameters of the attacker, but also on communications, available for the designed devices, their abstract elements and sub-elements as well security elements. Let's consider each dependence in more detail.

The communication levels that are possible for the designed devices are stored in the *communication_levels* table of the database. This table represents levels of communication with the help of a unique identificator and description. The content of the table is based on the architecture of microcontroller-based physical security systems provided in Section 1.3, see Figure 43.

	id	name
1	1	component to controller
2	2	controller to component
3	3	controller to controller
4	4	device to device

Figure 43. Content of the database: communication levels

The connections between levels of communication and possibility to implement attack actions are stored in the *communication_and_actions* table of the database. The content of the table is unique combinations of identificators from the following tables: *communication_levels* and *attack_actions*, see Figure 44.

	id	type_id	action_id
1	1	1	8
2	2	2	8
3	3	3	8
4	4	4	9
5	5	4	10
6	6	4	12

Figure 44. Content of the database: communication levels and actions

Based on such a table, it is possible to extract attack actions that are possible in accordance with levels of communication that are available for the designed device (for example, component to controller — 1, controller to component — 2 and controller to controller — 3), with the help of the following SQL sequence:

```
SELECT attack_actions.*
FROM attack_actions
WHERE id = ANY (
 SELECT DISTINCT action_id
 FROM communication_and_actions
 WHERE type_id IN(1, 2, 3)
) ORDER BY attack_actions.id;
```

The output of such an SQL sequence is equal to `(8, imw, "interception, modification or termination of wired communications")`.

In this work it was decided to not connect abstract elements and sub-elements to attack actions directly, because the number of possible elements is huge, while the number of their types is limited in accordance with the microcontroller-based physical security systems attack surface. Possible types of elements are stored in the *elements_types* table of the database. This table represents types of elements with the help of a unique identificator and description, see Figure 45.

	id	name
1	1	environment sensors
2	2	monitoring sensors
3	3	environment receivers
4	4	electronic component
5	5	microcontroller with rewritable firmware
6	6	microcontroller with rewritable bootloader
7	7	microcontroller with update system
8	8	device with wireless interface
9	9	device with sleep mode
10	10	system with users
11	11	system that relies on power grid
12	12	system with web-services
13	13	system with database

Figure 45. Content of the database: types of elements

The connections between types of elements and possibility to implement attack actions are stored in the *elements_and_actions* table of the database. The content of the table is unique combinations of identificators from the following tables: *elements_types* and *attack_actions*, see Figure 46.

	id	type_id	action_id
1	1	1	1
2	2	2	2
3	3	3	1
4	4	4	3
5	5	4	4
6	6	5	5
7	7	6	6
8	8	7	7
9	9	8	11
10	10	9	8
11	11	10	13
12	12	11	14
13	13	12	15
14	14	13	16

Figure 46. Content of the database: types of elements and actions

Based on such a table, it is possible to extract attack actions that are possible in accordance with abstract elements and sub-elements of the designed device based on their types (for example, environment sensors — 1, monitoring sensors — 2, microcontroller with rewritable firmware — 5 and device with wireless interface — 8), with the help of the following SQL sequence:

```
SELECT attack_actions.*
FROM attack_actions
WHERE id = ANY(
 SELECT DISTINCT action_id
 FROM elements_and_actions
 WHERE type_id IN(1, 2, 5, 8)
) ORDER BY attack_actions.id;
```

The output of such an SQL sequence is presented in Figure 47.

	id	name	description
1	1	gie	generation of incorrect component events
2	2	bcd	bypassing component detection algorithms
3	5	rfw	replacement of the firmware
4	11	iec	increased energy consumption

Figure 47. SQL sequence: attack actions that are possible based on elements

Security elements are stored in the *security_elements* table of the database. This table represents security elements with the help of a unique identificator and description. The content of the table is based on the security elements model provided in Chapter 3, see Figure 48.

	id	name
1	1	anomaly detection algorithm
2	2	hidden placement of sensors
3	3	events correlation algorithm
4	4	vandal-proof device case
5	5	hardware authentication
6	6	firmware encryption
7	7	bootloader encryption
8	8	removement of physical update interface
9	9	data encryption
10	10	mutual authentication
11	11	strong login credentials
12	12	password policy
13	13	brute-force protection
14	14	secure key distribution mechanism

	id	name
15	15	behaviour-based anomaly detection
16	16	devices isolation / limitation
17	17	training of operators and users
18	18	security policy
19	19	uninterruptible power supplies
20	20	backup power supply
21	21	firewall
22	22	update mechanism
23	23	backup mechanism
24	24	logging mechanism
25	25	input validation
26	26	strict access policy
27	27	separate users for different operations

Figure 48. Content of the database: security elements

The connections between security elements and possibility to implement attack actions are stored in the *security_and_actions* table of the database. The content of the table is unique combinations of identificators from the following tables: *security_elements* and *attack_actions*, see Figure 49.

	id	element_id	action_id
1	1	1	1
2	2	2	1
3	3	3	2
4	4	2	2
5	5	4	3
6	6	5	3
7	7	4	4
8	8	4	5
9	9	6	5
10	10	4	6
11	11	7	6
12	12	4	7
13	13	8	7
14	14	4	8
15	15	9	8
16	16	10	8
17	17	11	9
18	18	12	9
19	19	13	9

	id	element_id	action_id
20	20	9	10
21	21	14	10
22	22	15	11
23	23	16	11
24	24	9	12
25	25	11	12
26	26	14	12
27	27	17	13
28	28	18	13
29	29	19	14
30	30	20	14
31	31	21	15
32	32	22	15
33	33	23	15
34	34	24	15
35	35	25	16
36	36	26	16
37	37	27	16
38	38	11	16

Figure 49. Content of the database: security elements and actions

Based on such a table, it is possible to extract security elements that are required to integrate into the designed devices to prevent attack actions that are possible in accordance with it elements and levels of communication as well as parameters of the attacker (for example, generation of incorrect component events — 1, replacement of the firmware — 5, malfunction of the update system — 7, increased energy consumption — 11 and interception, modification or termination of wireless communications — 12), with the help of the following SQL sequence:

```
SELECT security_elements.*
FROM security_elements
WHERE id = ANY(
 SELECT element_id
 FROM security_and_actions
 WHERE action_id IN(1, 5, 7, 11, 12)
) ORDER BY security_elements.id;
```

The output of such an SQL sequence is presented in Figure 50.

	id	name
1	1	anomaly detection algorithm
2	2	hidden placement of sensors
3	4	vandal-proof device case
4	6	firmware encryption
5	8	removement of physical update interface
6	9	data encryption
7	11	strong login credentials
8	14	secure key distribution mechanism
9	15	behaviour-based anomaly detection
10	16	devices isolation / limitation

Figure 50. SQL sequence: security elements to prevent attack actions

5.2.2. Storage of tasks, abilities and requirements

The structure of the database for this part also contains 11 tables, see Figure 51.

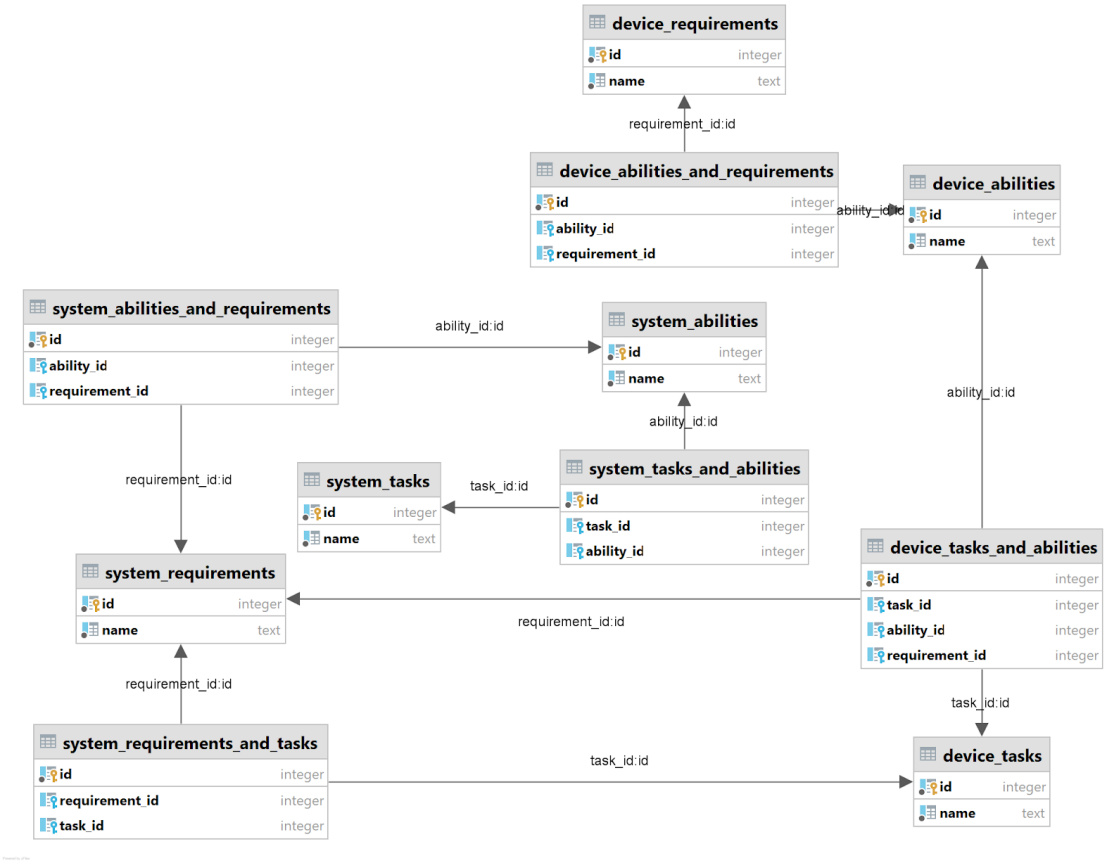

Figure 51. Database structure: tasks, abilities and requirements

Tasks, abilities and requirements are divided into the system and devices ones. Such a division is based on data structures, presented in Section 4.1. According to the algorithm for the formation of requirements for the system, tasks of the system are linked with abilities of the system, abilities of the system — with requirements of the system, requirements for the system — with tasks of devices, tasks of devices — with abilities of devices and abilities devices with requirements for devices. Let's consider database tables representing those tasks, abilities and requirements as well as connections between them in more detail.

Tasks of the designed system are stored in the *system_tasks* table of the database. This table represents tasks of the system with the help of a unique identificator and description, see Figure 52.

	id	name
1	1	centralized system management
2	2	static perimeter monitoring
3	3	mobile perimeter monitoring

Figure 52. Content of the database: tasks of the system

Abilities of the designed system are stored in the *system_abilities* table of the database. This table represents abilities of the system with the help of a unique identificator and description, see Figure 53.

	id	name
1	1	to store and process system data
2	2	to run executable applications
3	3	to download and install software updates
4	4	to create wireless access points
5	5	to communicate with mobile robots of the system
6	6	to communicate with charging stations of the system
7	7	to provide user interface for operators of the system

	id	name
8	8	to provide wireless charging
9	9	to monitor the perimeter nearby
10	10	to communicate with the server of the system
11	11	to be charged in a wireless way
12	12	to navigate through the perimeter
13	13	to detect and chase intruders

Figure 53. Content of the database: abilities of the system

Connections between tasks and abilities of the system are stored in the *system_tasks_and_abilities* table of the database. The content of the table is unique combinations of identificators from the following tables: *system_tasks* and *system_abilities*, see Figure 54.

	id	task_id	ability_id
1	1	1	1
2	2	1	2
3	3	1	3
4	4	1	4
5	5	1	5
6	6	1	6
7	7	1	7
8	8	2	8

	id	task_id	ability_id
9	9	2	9
10	10	2	5
11	11	2	10
12	12	3	11
13	13	3	12
14	14	3	13
15	15	3	6
16	16	3	10

Figure 54. Content of the database: tasks and abilities of the system

Based on such a table, it is possible to extract abilities that the designed system should have in accordance with tasks that the designed system should perform (for example, static perimeter monitoring — 2, mobile perimeter monitoring — 3), with the help of the following SQL sequence:

```
SELECT system_abilities.*
FROM system_abilities
WHERE id = ANY(
 SELECT DISTINCT ability_id
 FROM system_tasks_and_abilities
 WHERE task_id IN(2,3)
) ORDER BY id;
```

The output of such an SQL sequence is presented in Figure 55.

	id	name
1	5	to communicate with mobile robots of the system
2	6	to communicate with charging stations of the system
3	8	to provide wireless charging
4	9	to monitor the perimeter nearby

	id	name
5	10	to communicate with the server of the system
6	11	to be charged in a wireless way
7	12	to navigate through the perimeter
8	13	to detect and chase intruders

Figure 55. SQL sequence: abilities of the system

Requirements for the designed system are stored in the *system_requirements* table of the database. This table represents requirements of the system with the help of a unique identificator and description, see Figure 56.

	id	name
1	1	device that represents the server of the system
2	2	device that represents the charging stations of the system
3	3	device that represents the mobile robots of the system

Figure 56. Content of the database: requirements for the system

Connections between the system abilities and requirements are stored in the *system_abilities_and_requirements* table of the database. The content of the table is unique combinations of identificators from the following tables: *system_abilities* and *system_requirements*, see Figure 57.

	id	ability_id	requirement_id
1	1	1	1
2	2	2	1
3	3	3	1
4	4	4	1
5	5	5	1
6	6	6	1
7	7	7	1
8	8	8	2

	id	ability_id	requirement_id
9	9	9	2
10	10	5	2
11	11	10	2
12	12	11	3
13	13	12	3
14	14	13	3
15	15	6	3
16	16	10	3

Figure 57. Content of the database: system abilities and requirements

Based on such a table, it is possible to extract requirements for the designed system in accordance with abilities that the designed system should have (for example, to navigate through the perimeter — 12, to detect and chase intruders — 13), with the help of the following SQL sequence:

```
SELECT system_requirements.*
FROM system_requirements
WHERE id = ANY(
 SELECT DISTINCT requirement_id
 FROM system_abilities_and_requirements
 WHERE ability_id IN(12, 13)
) ORDER BY id;
```

The output of such an SQL sequence is the following requirement: `(3, “device that represents the mobile robots of the system”)`.

Tasks of designed devices are stored in the *device_tasks* table of the database. This table represents tasks of devices with the help of a unique identificator and description, see Figure 58.

	id	name
1	1	work cycle support
2	2	interaction with operators
3	3	interaction with other devices
4	4	navigation through the perimeter
5	5	interaction with intruders
6	6	interaction with charging stations
7	7	interaction with the server
8	8	interaction with mobile robots

Figure 58. Content of the database: tasks of devices

Connections between requirements for the system and tasks of devices are stored in the *system_requirements_and_tasks* table of the database. The content of the table is unique combinations of identificators from the following tables: *system_requirements* and *device_tasks*, see Figure 59.

	id	requirement_id	task_id
1	1	1	1
2	2	1	2
3	3	1	3
4	4	2	1
5	5	2	5
6	6	2	8

	id	requirement_id	task_id
7	7	2	7
8	8	3	1
9	9	3	4
10	10	3	5
11	11	3	6
12	12	3	7

Figure 59. Content of the database: system requirements and devices tasks

Based on such a table, it is possible to extract tasks of devices in accordance with requirements for the designed system (for example, device that represents the mobile robots of the system — 3), with the help of the following SQL sequence:

```
SELECT device_tasks.*
FROM device_tasks
WHERE id = ANY(
 SELECT DISTINCT task_id
 FROM system_requirements_and_tasks
 WHERE requirement_id IN(3)
) ORDER BY id;
```

The output of such an SQL sequence is presented in Figure 60.

	id	name
1	1	work cycle support
2	4	navigation through the perimeter
3	5	interaction with intruders
4	6	interaction with charging stations
5	7	interaction with the server

Figure 60. SQL sequence: tasks of devices

Abilities of designed devices are stored in the *device_abilities* table of the database. This table represents abilities of devices with the help of a unique identificator and description, see Figure 61.

	id	name
1	1	to store data
2	2	to update software
3	3	to run applications
4	4	to create wireless access points
5	5	to provide graphical user interface
6	6	to communicate with other devices
7	7	to update firmware
8	8	to be charged in a wireless way
9	9	to move

	id	name
10	10	to avoid obstacles
11	11	to navigate
12	12	to detect intruders
13	13	to chase intruders
14	14	to park near charging stations
15	15	to communicate with the server
16	16	to charge parked devices
17	17	to help mobile robots to park near

Figure 61. Content of the database: abilities of devices

Connections between tasks and abilities devices are stored in the *device_tasks_and_abilities* table of the database. The content of the table is unique combinations of identificators from the following tables: *device_tasks*, *device_abilities* and *system_requirements*, see Figure 62.

	id	task_id	ability_id	requirement_id
1	1	1	1	1
2	2	1	2	1
3	3	1	3	1
4	4	1	4	1
5	5	2	5	1
6	6	3	6	1
7	7	1	7	2
8	8	1	16	2
9	9	5	12	2
10	10	8	17	2

	id	task_id	ability_id	requirement_id
11	11	7	15	2
12	12	1	7	3
13	13	1	8	3
14	14	4	9	3
15	15	4	10	3
16	16	4	11	3
17	17	5	12	3
18	18	5	13	3
19	19	6	14	3
20	20	7	15	3

Figure 62. Content of the database: tasks and abilities of devices

Based on such a table, it is possible to extract abilities of devices in accordance with tasks of devices and requirements for the system (for example, requirements: device that represents the charging stations — 3; tasks: work cycle support — 1, interaction with intruders — 5), with the help of the following SQL sequence:

```
SELECT device_abilities.*
FROM device_abilities
WHERE id = ANY(
 SELECT DISTINCT ability_id
 FROM device_tasks_and_abilities
 WHERE requirement_id IN(2)
 AND task_id IN(1, 5)
) ORDER BY id;
```

The output of such an SQL sequence is presented in Figure 63.

	id	name
1	7	to update firmware
2	12	to detect intruders
3	16	to charge parked devices

Figure 63. SQL sequence: abilities of devices

Requirements for devices are stored in the *device_requirements* table of the database. This table represents requirements for devices with the help of a unique identificator and description, see Figure 64.

	id	name
1	1	32-bit operating system
2	2	sql database
3	3	wire network interface
4	4	software update server
5	5	software update mechanism
6	6	wireless network interface
7	7	access point configuration mechanism
8	8	application with graphical user interface
9	9	application-database connection
10	10	data processing algorithm
11	11	data presentation algorithm
12	12	devices communication algorithm
13	13	bootloader
14	14	firmware update mechanism
15	15	wireless charge receiver
16	16	battery
17	17	charge monitoring algorithm
18	18	collector motor
19	19	movement algorithm

	id	name
20	20	distance sensor
21	21	touch sensor
22	22	servo drive
23	23	obstacles detection algorithm
24	24	obstacles avoidance algorithm
25	25	encoder
26	26	map construction algorithm
27	27	path construction algorithm
28	28	motion sensor
29	29	noise sensor
30	30	intruders detection algorithm
31	31	intruders chase algorithm
32	32	wireless signal receiver
33	33	parking algorithm
34	34	server communication algorithm
35	35	wireless charge transmitter
36	36	wireless signal transmitter
37	37	parking direction algorithm

Figure 64. Content of the database: requirements for devices

Connections between devices abilities and requirements are stored in the *device_abilities_and_requirements* table of the database. The content of the table is unique combinations of identificators from the following tables: *device_abilities* and *device_requirements*, see Figure 65.

	id	ability_id	requirement_id
1	1	1	1
2	2	1	2
3	3	2	3
4	4	2	4
5	5	2	5
6	6	3	1
7	7	4	1
8	8	4	6
9	9	4	7
10	10	5	8
11	11	5	9
12	12	5	10
13	13	5	11
14	14	6	6
15	15	6	12
16	16	7	6
17	17	7	13
18	18	7	14
19	19	8	15
20	20	8	16
21	21	8	17
22	22	9	18

	id	ability_id	requirement_id
23	23	9	19
24	24	10	20
25	25	10	21
26	26	10	22
27	27	10	23
28	28	10	24
29	29	11	25
30	30	11	26
31	31	11	37
32	32	12	27
33	33	12	28
34	34	12	22
35	35	12	29
36	36	13	20
37	37	13	30
38	38	14	31
39	39	14	32
40	40	15	6
41	41	15	33
42	42	16	34
43	43	17	35
44	44	17	36

Figure 65. Content of the database: devices abilities and requirements

Based on such a table, it is possible to extract requirements for devices in accordance with their abilities (for example, to update firmware — 7, to detect intruders — 12), with the help of the following SQL sequence:

```
SELECT device_requirements.*
FROM device_requirements
WHERE id = ANY(
 SELECT DISTINCT requirement_id
 FROM device_abilities_and_requirements
 WHERE ability_id IN(7, 12)
) ORDER BY id;
```

The output of such an SQL sequence is presented in Figure 66.

	id	name
1	6	wireless network interface
2	13	bootloader
3	14	firmware update mechanism
4	22	servo drive
5	27	path construction algorithm
6	28	motion sensor
7	29	noise sensor

Figure 66. SQL sequence: abilities of devices

5.2.3. Storage of abstract elements, sub-elements and links

The structure of the database for this part contains 25 tables. For ease of understanding, it was decided to divide this part into sub-parts: abstract (3.1) elements, (3.2) sub-elements, (3.3) links and (3.4) recommendations.

The abstract elements sub-part of the database contains 11 tables, see Figure 67.

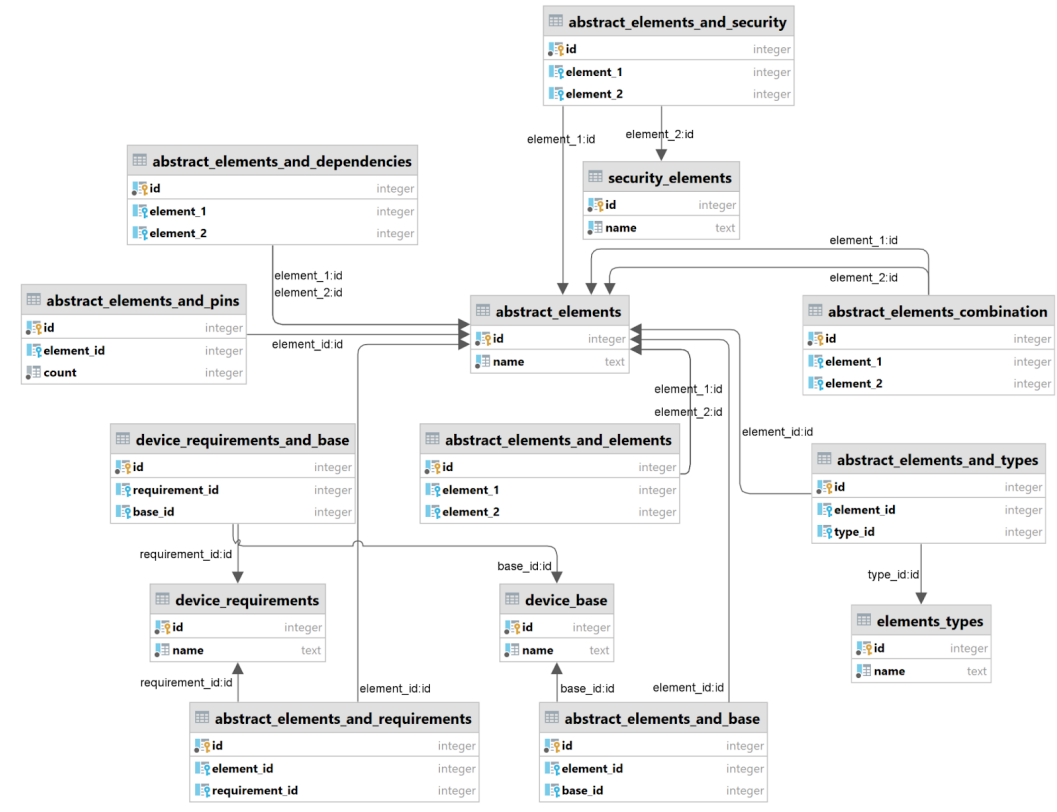

Figure 67. Database structure: abstract elements

It is important to note that the third part of the database structure is related to the storage of data that is required for the design of the abstract model of the system. This data is used by the algorithms, presented in Sections 4.2 and 4.3.

Abstract elements of the designed system are stored in the *abstract_elements* table of the database. This table represents elements with the help of a unique identificator and name, see Figure 68.

	id	name
1	1	32-bit operating system
2	2	sql database
3	3	application with graphical user interface
4	4	wireless charge receiver
5	5	battery
6	6	collector motor
7	7	distance sensor
8	8	touch sensor
9	9	servo drive
10	10	motion sensor
11	11	noise sensor
12	12	wireless signal receiver
13	13	wireless charge transmitter

	id	name
14	14	wireless signal transmitter
15	15	encoder
16	16	one-board computer
17	17	microcontroller for electronic components
18	18	microcontroller for wireless communication
19	19	firmware for electronic components
20	20	firmware for wireless communication
21	21	micro SD
22	22	motor shield
23	23	troyka shield
24	24	vandal-proof device case
25	25	wireless access point

Figure 68. Content of the database: abstract elements of the system

The extraction of abstract elements of devices of the designed system is possible in accordance with security elements (*security_elements* and *elements_types* tables from part 1), requirements for devices (*device_requirements* table from part 2), bases of devices (*device_base* table from this part of the database) and already extracted elements. Let's consider each extraction possibility in more detail.

Connections between abstract and security elements are stored in the *abstract_elements_and_security* table of the database. The content of the table is unique combinations of identificators from the following tables: *abstract_elements* and *security_elements*, see Figure 69.

	id	element_1	element_2
1	1	24	4
2	2	5	19
3	3	5	20

Figure 69. Content of the database: abstract and security elements

Based on such a table, it is possible to extract abstract elements that are representing corresponding security elements (for example, uninterruptible power supply — 19), with the help of the following SQL sequence:

```
SELECT abstract_elements.*
FROM abstract_elements
WHERE id = ANY(
 SELECT element_1
 FROM abstract_elements_and_security
 WHERE element_2 IN(19)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: `(5, “battery”)`.

Connections between abstract elements and requirements for designed devices are stored in the *abstract_elements_and_requirements* table of the database. The content of the table is unique combinations of identificators from the following tables: *abstract_elements* and *device_requirements*, see Figure 70.

	id	element_id	requirement_id
1	1	1	1
2	2	2	2
3	3	3	8
4	4	4	15
5	5	5	16
6	6	6	18
7	7	7	20
8	8	8	21

	id	element_id	requirement_id
9	9	9	22
10	10	10	27
11	11	11	28
12	12	12	31
13	13	13	34
14	14	14	35
15	15	15	37
16	16	25	7

Figure 70. Content of the database: abstract elements and requirements

Based on such a table, it is possible to extract abstract elements that are required to satisfy requirements for designed devices (for example, access point configuration mechanism — 7), with the help of the following SQL sequence:

```
SELECT abstract_elements.*
FROM abstract_elements
WHERE id = ANY(
 SELECT element_id
 FROM abstract_elements_and_requirements
 WHERE requirement_id IN(7)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: `(25, “wireless access point”)`.

Bases of the designed devices are stored in the *device_base* table of the database. This table represents bases with the help of a unique id and name, see Figure 71.

	id	name
1	1	one-board computer
2	2	connected microcontrollers
3	3	microcontroller

Figure 71. Content of the database: bases of devices

Connections between requirements for devices of the designed system and their bases are stored in the *device_requirements_and_base* table of the database. The content of the table is unique combinations of identificators from the following tables: *device_requirements* and *device_base*, see Figure 72.

	id	requirement_id	base_id
1	1	1	1
2	2	2	1
3	3	3	2
4	4	4	1
5	5	5	1
6	6	6	2
7	7	7	2
8	8	8	1
9	9	9	1
10	10	10	3
11	11	11	3
12	12	12	3
13	13	13	3
14	14	14	3
15	15	15	3
16	16	16	3
17	17	17	3
18	18	18	3
19	19	19	3

	id	requirement_id	base_id
20	20	20	3
21	21	21	3
22	22	22	3
23	23	23	3
24	24	24	3
25	25	25	3
26	26	26	3
27	27	27	3
28	28	28	3
29	29	29	3
30	30	30	3
31	31	31	3
32	32	32	3
33	33	33	3
34	34	34	3
35	35	35	3
36	36	36	3
37	37	37	3

Figure 72. Content of the database: requirements for devices and their bases

Based on such a table, it is possible to extract the base of the devices in accordance with its requirements (for example, firmware update mechanism — 14, servo drive — 22, server communication algorithm — 34), with the help of the following sequence:

```
SELECT device_base.*
FROM device_base
WHERE device_base.id = (
 SELECT MIN(base_id)
 FROM device_requirements_and_base
 WHERE requirement_id IN(14, 22, 34)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`3`, “`microcontroller`”).

Connections between abstract elements and bases of devices are stored in the *abstract_elements_and_base* table of the database. The content of the table is unique combinations of identificators from the following tables: *abstract_elements* and *device_base*, see Figure 73.

	id	element_id	base_id
1	1	16	1
2	2	17	2
3	3	18	2
4	4	19	2
5	5	20	2
6	6	17	3
7	7	19	3

Figure 73. Content of the database: requirements for devices and their bases

Based on such a table, it is possible to extract abstract elements of devices that are representing its base (for example, microcontroller — 3), with the help of the following SQL sequence:

```
SELECT abstract_elements.*
FROM abstract_elements
WHERE id = ANY(
 SELECT element_id
 FROM abstract_elements_and_base
 WHERE base_id = 3
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`17`, “`microcontroller for electronic components`”), (`19`, “`firmware for electronic components`”).

Connections between abstract elements and their types are stored in the *abstract_elements_and_types* table of the database. The content of the table is unique combinations of identificators from the following tables: *abstract_elements* and *elements_types*, see Figure 74.

	id	element_id	type_id
1	1	7	1
2	2	8	1
3	3	15	1
4	4	10	2
5	5	11	2
6	6	12	3
7	7	4	4
8	8	5	4
9	9	6	4
10	10	7	4
11	11	8	4
12	12	9	4
13	13	10	4
14	14	11	4
15	15	12	4
16	16	13	4
17	17	14	4

	id	element_id	type_id
18	18	15	4
19	19	21	4
20	20	22	4
21	21	23	4
22	22	17	5
23	23	18	5
24	24	17	6
25	25	18	6
26	26	16	7
27	27	17	7
28	28	18	7
29	29	16	8
30	30	18	8
31	31	3	10
32	32	16	11
33	33	13	11
34	34	2	13

Figure 74. Content of the database: abstract elements and their types

Based on such a table, it is possible to extract types of abstract elements of the designed device to check the attack actions that are possible based on them (for example, collector motor – 6, distance sensor – 7, wireless signal transmitter – 14), with the help of the following SQL sequence:

```
SELECT elements_types.*
FROM elements_types
WHERE id = ANY(
 SELECT type_id
 FROM abstract_elements_and_types
 WHERE element_id IN(6, 7, 14)
);
```

The output of such an SQL sequence is as follows: (1, “environment sensor”), (4, “electronic component”).

Connections between abstract elements are stored in the database in the *abstract_elements_and_elements* table. The content of the table is unique combinations of identificators from the *abstract_elements* table, see Figure 75.

	id	element_1	element_2
1	1	16	21
2	2	6	22
3	3	17	23

Figure 75. Content of the database: abstract elements with itself

The content of this table states, for example, that if we already have an element (16, “one-board computer”) then it is required to add another one (21, “micro-SD”) to the abstract system composition.

Possibilities to combine abstract elements together are stored in the database in the *abstract_elements_combination* table. The content of the table is also unique combinations of identificators from the *abstract_elements* table, see Figure 76.

	id	element_1	element_2
1	1	1	2
2	2	1	3
3	3	1	25
4	4	5	4
5	5	5	13
6	6	16	21
7	7	16	5
8	8	17	22
9	9	17	23
10	10	17	19
11	11	17	5
12	12	18	20
13	13	21	1
14	14	22	6

	id	element_1	element_2
15	15	22	23
16	16	22	5
17	17	23	7
18	18	23	8
19	19	23	9
20	20	23	10
21	21	23	11
22	22	23	12
23	23	23	14
24	24	23	15
25	25	23	18
26	26	24	16
27	27	24	17

Figure 76. Content of the database: abstract elements combination

The content of this table states, for example, that an element (21, “micro-SD”) can be combined with an element (16, “one-board computer”) and be its sub-component during the construction of devices components composition.

Dependencies between abstract elements are stored in the database in the *abstract_elements_and_dependencies* table. The content of the table is unique combinations of identificators from the *abstract_elements* table, see Figure 77.

The content of this table states, for example, that available for the selection options of the element (23, “troyka shield”) are depending on the selection of the element (17, “microcontroller for electronic components”), see Section 4.4.

	id	element_1	element_2
1	1	1	2
2	2	1	3
3	3	5	4
4	4	5	13
5	5	16	1
6	6	16	21
7	7	16	5
8	8	17	22
9	9	17	23
10	10	17	5
11	11	22	6

	id	element_1	element_2
12	12	22	5
13	13	23	7
14	14	23	8
15	15	23	9
16	16	23	10
17	17	23	12
18	18	23	18
19	19	23	11
20	20	23	15
21	21	23	14

Figure 77. Content of the database: abstract elements dependencies

Connections between abstract elements of devices of the designed system and number of digital and analog pins, required for their connection to controllers are stored in the *abstract_elements_and_pins* table of the database, see Figure 78.

	id	element_id	count
1	1	7	1
2	2	8	1
3	3	9	1
4	4	10	1
5	5	11	1
6	6	15	1
7	7	22	4
8	8	12	1
9	9	14	1
10	10	18	4

Figure 78. Content of the database: abstract elements and pins

The content of this table states, for example, that an element (`22, “motor shield”`) requires 4 pins of the controller to be connected. This requirement in combination with requirements of other elements is used to form the requirement for a minimal number of pins of the controllers used in the designed device.

The abstract sub-elements part of the database contains 5 tables, see Figure 79.

Abstract sub-elements of the designed system are stored in the *abstract_subelements* table of the database. This table represents sub-elements with the help of a unique identificator and name.

The extraction of abstract sub-elements of devices of the designed system is possible in accordance with security elements (*security_elements* table from part 1), requirements for devices (*device_requirements* table from part 2), bases of devices (*device_base* table from part 3.1) and abstract elements (*abstract_elements* table from part 3.1). Let’s consider them in more detail.

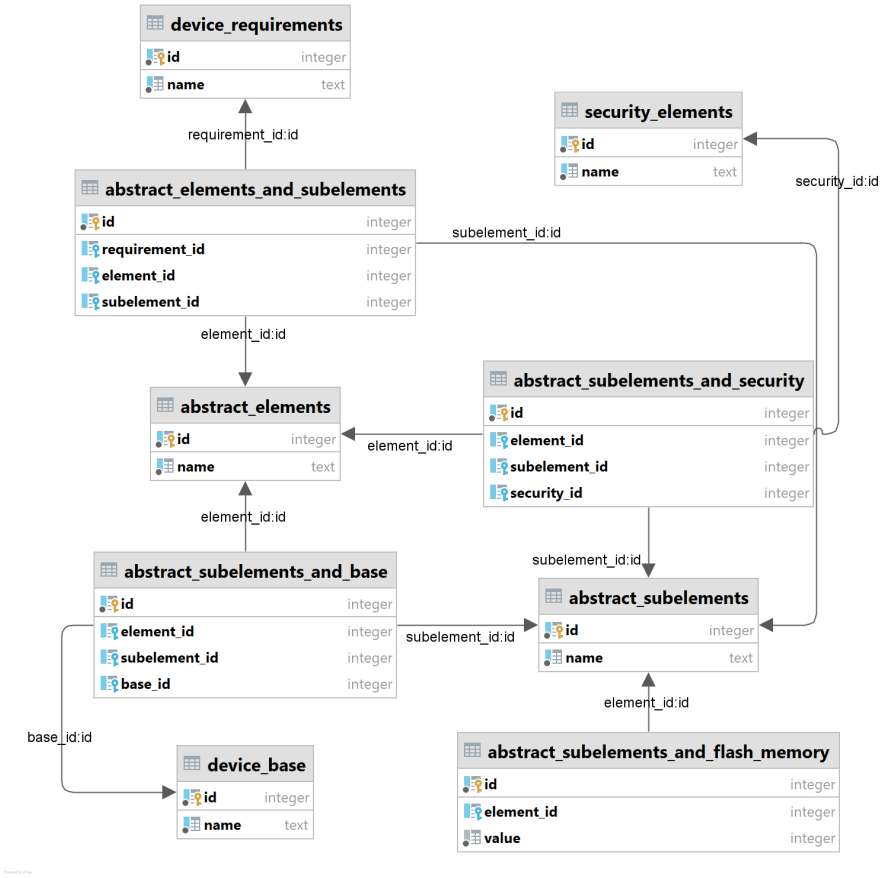

Figure 79. Database structure: abstract sub-elements

The content of the *abstract_subelements* table is presented in Figure 80.

id	name
1	wire network interface
2	software update server
3	software update mechanism
4	wireless network interface
5	access point configuration mechanism
6	application-database connection
7	data processing algorithm
8	data presentation algorithm
9	devices communication algorithm
10	bootloader
11	firmware update mechanism
12	charge monitoring algorithm
13	movement algorithm
14	obstacles detection algorithm
15	obstacles avoidance algorithm
16	map construction algorithm
17	path construction algorithm
18	intruders detection algorithm
19	intruders chase algorithm
20	parking algorithm
21	server communication algorithm
22	parking direction algorithm
23	microcontrollers communication algorithm

	id	name
24	24	electronic components interaction algori…
25	25	sensors anomaly detection algorithm
26	26	data collection algorithm for anomaly de…
27	27	events correlation algorithm
28	28	hardware authentication algorithm
29	29	firmware encryption algorithm
30	30	firmware decryption algorithm
31	31	bootloader encryption algorithm
32	32	bootloader decryption algorithm
33	33	communication data encryption and decryp…
34	34	mutual authentication algorithm
35	35	strong login credentials
36	36	secure password policy
37	37	brute-force protection algorithm
38	38	secure key distribution mechanism
39	39	devices behaviour-based anomaly detectio…
40	40	devices isolation / limitation algorithm
41	41	firewall
42	42	logging mechanism
43	43	backup mechanism
44	44	input data validation algorithm
45	45	strict access policy
46	46	separate users for different operations

Figure 80. Content of the database: abstract sub-elements

Connections between abstract elements and sub-elements of devices are stored in the *abstract_elements_and_subelements* table of the database. The content of the table is unique combinations of ids from the following tables: *device_requirements*, *abstract_elements* and *abstract_subelement*, see Figure 81.

	id	requirement_id	element_id	subelement_id
1	1	3	16	1
2	2	4	1	2
3	3	5	1	3
4	4	6	16	4
5	5	7	1	5
6	6	9	3	6
7	7	10	3	7
8	8	11	3	8
9	9	12	3	9
10	10	13	17	10
11	11	13	18	10
12	12	14	19	11
13	13	14	20	11

	id	requirement_id	element_id	subelement_id
14	14	17	19	12
15	15	19	19	13
16	16	23	19	14
17	17	24	19	15
18	18	25	19	16
19	19	26	19	17
20	20	29	19	18
21	21	30	19	19
22	22	32	19	20
23	23	33	19	21
24	24	33	20	21
25	25	36	19	22

Figure 81. Content of the database: abstract elements and sub-elements

Based on such a table, it is possible to extract sub-elements of abstract elements in accordance with requirements for designed devices (for example, element: firmware update mechanism — 14, charge monitoring algorithm — 17, movement algorithm — 19), with the help of the following SQL sequence:

```
SELECT abstract_subelements.*
FROM abstract_subelements
WHERE id = ANY(
 SELECT subelement_id
 FROM abstract_elements_and_subelements
 WHERE element_id = 19
 AND requirement_id IN (14, 17, 19)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`11`, “`firmware update mechanism`”), (`12`, “`charge monitoring`”) and (`13`, “`movement algorithm`”).

Connections between abstract sub-elements of devices and their bases are stored in the *abstract_subelements_and_base* table of the database. The content of the table is unique combinations of ids from the following tables: *abstract_elements*, *abstract_subelement* and *device_bases*, see Figure 82.

	id	element_id	subelement_id	base_id
1	1	19	23	2
2	2	20	23	2
3	3	19	24	2
4	4	19	24	3

Figure 82. Content of the database: bases of devices and abstract sub-elements

Based on such a table, it is possible to extract sub-elements of devices in accordance with abstract elements and bases (for example, element: firmware for electronic components — 19, base: connected microcontrollers — 2), with the help of the following SQL sequence:

```
SELECT abstract_subelements.*
FROM abstract_subelements
WHERE id = ANY(
 SELECT subelement_id
 FROM abstract_subelements_and_base
 WHERE element_id = 19
 AND base_id = 2
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`23, "controllers communication algorithm"`) and (`24, "components interaction algorithm"`).

Connections between abstract sub-elements of devices and security elements are stored in the *abstract_subelements_and_security* table of the database. The content of the table is unique combinations of ids from the following tables: *abstract_elements*, *abstract_subelements* and *security_elements*, see Figure 83.

	id	element_id	subelement_id	security_id
1	1	19	25	1
2	2	19	27	3
3	3	19	30	6
4	4	19	33	9
5	5	19	34	10
6	6	20	30	6
7	7	20	33	9
8	8	20	34	10
9	9	20	26	15
10	10	3	29	6
11	11	3	31	7
12	12	3	32	7
13	13	25	33	9

	id	element_id	subelement_id	security_id
14	14	25	35	1
15	15	25	37	1
16	16	25	38	1
17	17	1	41	2
18	18	1	3	2
19	19	1	46	2
20	20	1	42	2
21	21	2	35	1
22	22	2	43	2
23	23	2	44	2
24	24	2	45	2
25	25	3	39	1
26	26	3	40	1

Figure 83. Content of the database: security elements and abstract sub-elements

Based on such a table, it is possible to extract sub-elements of devices in accordance with abstract and security elements (for example, abstract element: firmware for electronic components — 19, security element: anomaly detection algorithm – 1, events correlation algorithm – 3, data encryption – 9), with the help of the following SQL sequence:

```
SELECT abstract_subelements.*
FROM abstract_subelements
WHERE id = ANY(
 SELECT subelement_id
 FROM abstract_subelements_and_security
 WHERE element_id = 19
 AND security_id IN (1, 3, 9)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (25, “sensors anomaly detection algorithm”), (27, “events correlation algorithm”) and (33, “communication data encryption and decryption algorithm”).

Connections between abstract sub-elements of devices and amount of flash memory required for their work on controllers are stored in the *abstract_subelements_and_flash_memory* table of the database, see Figure 84.

	id	element_id	value
1	1	11	20
2	2	12	10
3	3	13	20
4	4	14	20
5	5	15	30
6	6	16	30
7	7	17	40
8	8	18	20

	id	element_id	value
9	9	19	40
10	10	20	20
11	11	21	30
12	12	22	40
13	13	23	40
14	14	26	40
15	15	33	100
16	16	34	20

Figure 84. Content of the database: abstract sub-elements and flash-memory

The content of this table states, for example, that an element (33, “communication data encryption and decryption algorithm”) requires 100 KB of flash memory of the controller. This requirement in combination with requirements of other sub-elements is used to form the requirement for a minimal amount of flash memory of the controllers used in the designed device.

The abstract links sub-part of the database contains 5 tables, see Figure 85.

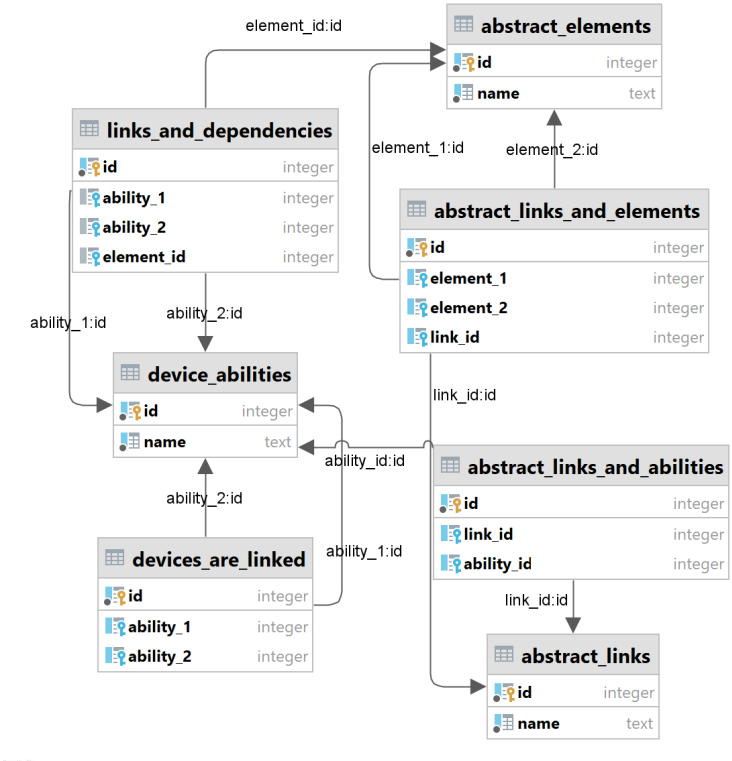

Figure 85. Database structure: abstract links

The extraction of abstract links between elements is possible in accordance with abstract elements (*abstract_elements* table from part 3.1) and abilities of devices (*device_abilities* table from part 2). Let's consider them in more detail.

Abstract links between elements of the designed system are stored in the *abstract_links* table of the database. The content of the table is based on the model of links between system elements, presented in Chapter 3. This table represents links with the help of a unique identificator and name, see Figure 86.

	id	name
1	1	pin-to-pin
2	2	SVG
3	3	VG
4	4	method
5	5	database
6	6	API
7	7	install

	id	name
8	8	flashing
9	9	wire
10	10	wireless
11	11	configure
12	12	slot
13	13	placement
14	14	TxRx

Figure 86. Content of the database: abstract links

Connections between abstract links between elements and abilities of devices are stored in the *abstract_links_and_abilities* table of the database. The content of the table is unique combinations of ids from the following tables: *abstract_links* and *abstract_abilities*, see Figure 87.

	id	link_id	ability_id
1	1	10	6
2	2	10	8
3	3	10	14
4	4	10	15
5	5	10	16
6	6	10	17

Figure 87. Content of the database: abstract links and device abilities

Based on such a table, it is possible to extract links between devices of the system in accordance with their abilities (for example, to park near charging stations — 14), with the help of the following SQL sequence:

```
SELECT abstract_links.*
FROM abstract_links
WHERE id = ANY(
 SELECT link_id
 FROM abstract_links_and_abilities
 WHERE ability_id IN (14)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`10, "wireless"`).

Possibility to decide if devices are linked together or not is based on their abilities and stored in the *devices_are_linked* table of the database. The content of the table is unique combinations of ids from *abstract_abilities* table, see Figure 88.

	id	name
1	1	to store data
2	2	to update software
3	3	to run applications
4	4	to create wireless access points
5	5	to provide graphical user interface
6	6	to communicate with other devices
7	7	to update firmware
8	8	to be charged in a wireless way
9	9	to move

	id	name
10	10	to avoid obstacles
11	11	to navigate
12	12	to detect intruders
13	13	to chase intruders
14	14	to park near charging stations
15	15	to communicate with the server
16	16	to charge parked devices
17	17	to help mobile robots to park near

Figure 88. Content of the database: devices are linked or not

Based on such a table, it is possible to decide if devices of the designed system are linked or not in accordance with their abilities (for example, device 1: to park near charging stations — 14, device 2: to help mobile robots to park near — 17), with the help of the following SQL sequence:

```
SELECT id
FROM devices_are_linked
WHERE ability_1 = 14
AND ability_2 = 17;
```

If the output of such an SQL sequence is not empty then there is a link.

Dependencies between abstract elements and links are stored in the database in the *links_and_dependencies* table. The content of the table is unique combinations of identificators from the *device_abilities* and *abstract_elements* tables, see Figure 89.

	id	ability_1	ability_2	element_id
1	1	6	15	16
2	2	6	15	18
3	3	6	15	25
4	4	15	6	16
5	5	15	6	18
6	6	15	6	25
7	7	8	16	4
8	8	8	16	5

	id	ability_1	ability_2	element_id
9	9	8	16	13
10	10	16	8	4
11	11	16	8	5
12	12	16	8	13
13	13	14	17	12
14	14	14	17	14
15	15	17	14	12
16	16	17	14	14

Figure 89. Content of the database: abstract links and dependencies

Based on such a table, it is possible to extract abstract elements, the selection of which are depending on the selection of abstract links between devices, while such links can be represented based on abilities of devices (for example, device 1: to park near charging stations — 14, device 2: to help mobile robots to park near — 17). The extraction is possible with the help of the following SQL sequence:

```
SELECT abstract_elements.*
FROM abstract_elements
WHERE id = ANY(
 SELECT element_id
 FROM links_and_dependencies
 WHERE ability_1 = 14
 AND ability_2 = 17
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (12, “wireless signal receiver”), (14, “wireless signal transmitter”).

Abstract links between abstract elements that occur in the process of their combination are stored in the *abstract_links_and_elements* table of the database. The content of the table is unique combinations of identificators from *abstract_elements* and *abstract_links* tables, see Figure 90.

	id	element_1	element_2	link_id
1	1	1	3	7
2	2	1	2	7
3	3	1	25	11
4	4	5	4	3
5	5	5	13	3
6	6	16	5	3
7	7	16	21	12
8	8	17	22	1
9	9	17	19	8
10	10	17	23	1
11	11	17	5	3
12	12	18	20	8
13	13	21	1	7
14	14	22	23	1

	id	element_1	element_2	link_id
15	15	22	6	3
16	16	22	5	3
17	17	23	18	14
18	18	23	15	2
19	19	23	12	2
20	20	23	11	2
21	21	23	10	2
22	22	23	7	2
23	23	23	9	2
24	24	23	8	2
25	25	23	14	2
26	26	24	16	13
27	27	24	17	13

Figure 90. Content of the database: abstract links between abstract elements

Based on such a table, it is possible to extract abstract links between abstract elements based on those elements (for example, element 1: single-board computer — 16, element 2: micro-SD — 21), with the help of the following SQL sequence:

```
SELECT abstract_links.*
FROM abstract_links
WHERE id = (
 SELECT link_id
 FROM abstract_links_and_elements
 WHERE element_1 = 16
 AND element_2 = 21
);
```

The output of such an SQL sequence is as follows: (12, “slot”). It means that it is required to slot a micro-SD card into a single-board computer to link them.

The recommendations sub-part of the database contains 4 tables, see Figure 91.

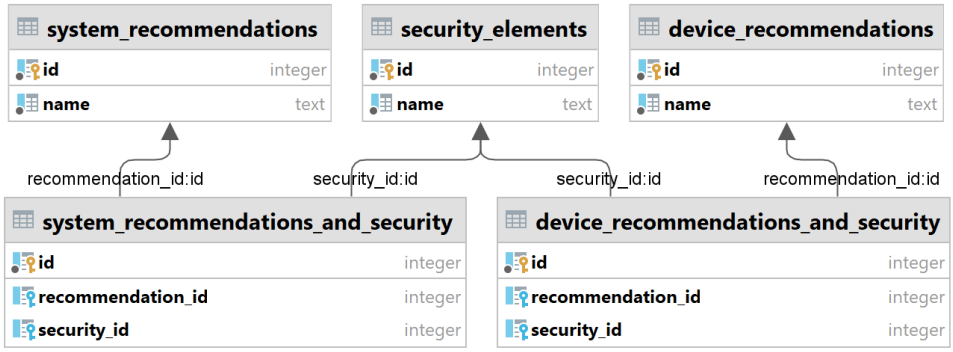

Figure 91. Database structure: recommendations

Security recommendations are divided into recommendations to the implementation of the system and to the implementation of its devices. In this work, recommendations are representing security elements that cannot be integrated into the system as an abstract element or sub-element, see Section 4.2.

Recommendations to the implementation of the designed system are stored in the *system_recommendations* table of the database. This table represents recommendations with the help of a unique identificator and name, see Figure 92.

	id	name
1	1	to develop and use strong password policy for all login credentials of the system
2	2	to educate operators and users of the system about social engineering attacks
3	3	to develop and use security policy regarding working with this system

Figure 92. Content of the database: recommendations to the system

Recommendations to the implementation of designed devices are stored in the *device_recommendations* table of the database. This table also represents recommendations with the help of a unique identificator and name, see Figure 93.

	id	name
1	1	to hide monitoring sensors of this device
2	2	to remove physical update interfaces from this device

Figure 93. Content of the database: recommendations to devices

Connections between recommendations to the implementation of the system and security elements are stored in the *system_recommendations_and_security* table of the database. The content of the table is unique combinations of ids from the following tables: *system_recommendations* and *security_elements*, see Figure 94.

	id	recommendation_id	security_id
1	1	1	12
2	2	2	17
3	3	3	18

Figure 94. Content of the database: system recommendations and security

Based on such a table, it is possible to extract recommendations to the system implementations in accordance with its security elements (for example, password policy — 12), with the help of the following SQL sequence:

```
SELECT system_recommendations.*
FROM system_recommendations
WHERE id = ANY (
 SELECT recommendation_id
 FROM system_recommendations_and_security
 WHERE security_id IN(12)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`1, “to develop and use strong password policy for all login credentials of the system”`).

Connections between recommendations to the implementation of devices and security elements are stored in the *device_recommendations_and_security* table of the database. The content of the table is unique combinations of ids from the following tables: *device_recommendations* and *security_elements*, see Figure 95.

	id	recommendation_id	security_id
1	1	1	2
2	2	2	8

Figure 95. Content of the database: devices recommendations and security

Based on such a table, it is possible to extract recommendations to devices implementations in accordance with their security elements (for example, hidden placement of sensors — 2), with the help of the following SQL sequence:

```
SELECT device_recommendations.*
FROM device_recommendations
WHERE id = ANY (
 SELECT recommendation_id
 FROM device_recommendations_and_security
 WHERE security_id IN(2)
) ORDER BY id;
```

The output of such an SQL sequence is as follows: (`1, “to hide monitoring sensors of this device”`).

5.2.4. Storage of detailed elements

The structure of the database for this part contains more than 50 tables. Moreover, in contrast to other parts, the number of tables in this part can only be increased by adding new components and controllers. This is due to the fact that this part of the database is related to concrete implementations that are represented as possible options for abstract elements during the detailing process of the abstract system model, see Section 4.4.

To represent the main idea of this part, it was decided, firstly, to show the sub-part of this part that is related to the connections between abstract elements and their concrete implementations and, secondly, to show an example of the database structure for one of the implementations.

The connections between abstract and detailed elements sub-part contains only 2 tables and is connected with *abstract_elements* table from part 2, see Figure 96.

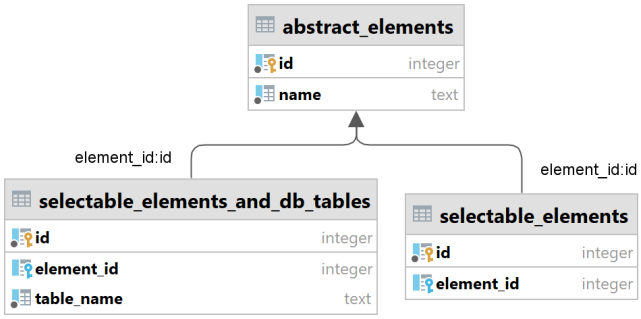

Figure 96. Database structure: abstract and detailed elements

Selectable elements are stored in the *selectable_elements* table of the database. Selectable for the abstract element means that it can be selected by the design methodology, see Section 4.5, because the database is filled with data about its concrete implementations. The table represents selectable elements with the help of a unique identificator and identificator of the abstract element, see Figure 97.

	id	element_id
1	1	1
2	2	2
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9
9	9	10
10	10	11

	id	element_id
11	11	12
12	12	13
13	13	14
14	14	15
15	15	16
16	16	17
17	17	18
18	18	21
19	19	22
20	20	23

Figure 97. Content of the database: abstract elements that are selectable

The content of this table states, for example, that an element (13, “wireless charge transmitter”) is selectable, while an element (3, “application with graphical user interface”) is not. It means that it would be necessary to develop such an application after the design of the system.

Database tables that are storing data about implementations of abstract elements are stored in the *selectable_elements_and_db_tables* table of the database. This structure helps the design methodology to navigate through the selection process during the detailing of the abstract system model, see Section 4.4. The table represents database tables with the help of a unique identificator, identificator of the abstract element and name of the database table, see Figure 98.

	id	element_id	table_name
1	1	1	operating_systems_32bit
2	2	2	sql_databases
3	3	4	wireless_charge_receivers
4	4	5	batteries
5	5	6	collector_motors
6	6	7	distance_sensors
7	7	8	touch_sensors
8	8	9	servo_drives
9	9	10	motion_sensors
10	10	11	noise_sensors

	id	element_id	table_name
11	11	12	wireless_signal_receivers
12	12	13	wireless_charge_transmitt…
13	13	14	wireless_signal_transmitt…
14	14	15	encoders
15	15	16	oneboard_computers
16	16	17	microcontrollers_for_elec…
17	17	18	mcs_for_ws_communication
18	18	21	micro_sds
19	19	22	motor_shields
20	20	23	troyka_shields

Figure 98. Content of the database: tables for implementations

The content of this table states, for example, that possible implementations of the abstract element (13, “wireless charge transmitter”) are stored in the *wireless_charge_transmitters* table of the database.

As an example of **the database structure for one of the implementations**, it was decided to show tables related to the abstract element (5, “battery”). This structure contains 13 tables, see Figure 99.

According to this structure, the selected implementation of the abstract element (5, “battery”) is connected with implementations of links between devices (*batteries_and_links_between_devices* table), microcontrollers for electronic components (*batteries_and_microcontrollers_for_electronic_components* table), motor shields (*batteries_and_motor_shields* table), one-board computers (*batteries_and_oneboard_computers* table), wireless charge receivers (*wireless_charge_receivers_and_batteries* table) and wireless charge transmitters (*wireless_charge_transmitters_and_batteries* table). All these connection tables are storing data about compatibility between implementations of abstract elements. In turn, tables for implementations are storing data about their parameters, providing a possibility to check their correspondence to the provided requirements.

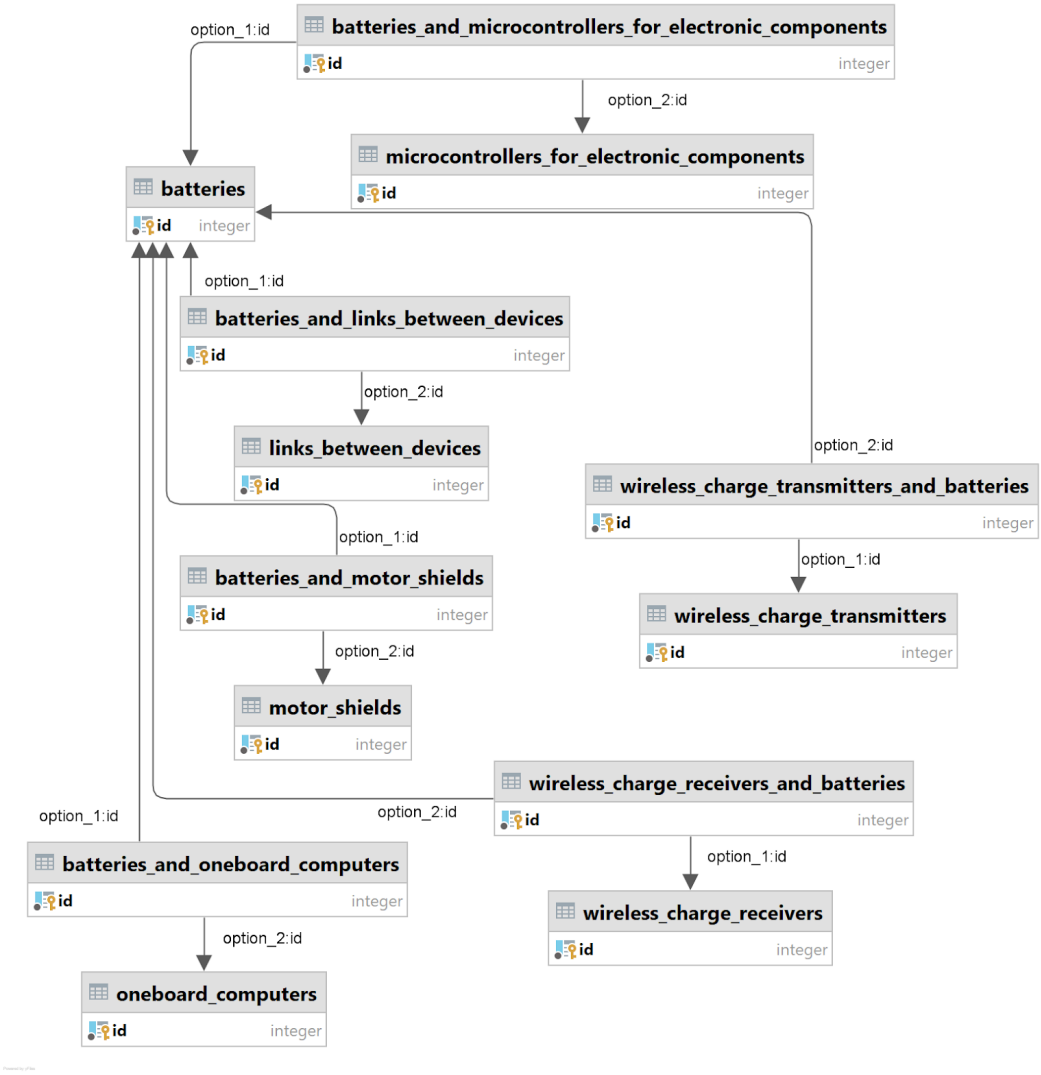

Figure 99. Database structure: detailed element example

The abstract element (`5`, “`battery`”) can have implementations with the following parameters: capacity in mAh, output current in A, length x width x height in mm, weight in g and price in rubles, see Figure 100.

	id	name	code	capa...	cur...	l...	w...	h...	w...	p...
1	1	Xiaomi M…	PLM07ZM	20000	3	73.5	153.5	27.5	440	2554
2	2	ZMI AURA…	QB822	20000	3	69.9	149.5	23.9	374	2618
3	3	Xiaomi M…	PLM12ZM	10000	3	71.2	147	14.2	223	1130
4	4	HIPER Fo…	100W	20000	3	70	140	20	500	5490
5	5	HIPER	MPX20000	20000	3	67	13	24.7	386	2990
6	6	Baseus A…	PPALL-LG…	20000	3	65	144	26	340	1690
7	7	Accessty…	10MPQP	10000	3	66	140	66	140	16
8	8	Power Ba…	AMP-B188	2000	0.6	55	33	20	80	1490
9	9	Power Sh…	AMP-B088	2000	0.7	69	53	20	100	1490

Figure 100. Content of the database: battery implementations

5.3. Script of the software implementation

The developed script contains more than 3000 lines of code (including comments) and works with such imports as psycopg2 [137], tkinter [138], pygubu [139], networkx [140], json [141], functools [142] and time [143].

The role of the script is to implement algorithms from Chapter 4, combine them together into the design methodology from Section 4.5 and to provide connections between the database from Section 5.2 and the interface from Section 5.4.

The script connects itself with the developed interface with the help of the pygubu library. Firstly, builder is created:

```
self.builder = builder = pygubu.Builder()
```

After that, the interface is loaded from the file:

```
builder.add_from_file("interface/design_GUI.ui")
```

This allows the script to get access to objects of the interface and control them: default state, selected values and callback functions of objects as well as links between them can be defined. For example, it is possible to create the main windows of the interface and run it:

```
# 3: Create the window
self.window = builder.get_object("frame1")

def run(self):
 self.window.mainloop()

app = DesignApp()
app.run()
```

The script connects itself with the developed database with the help of the psycopg2 library and its extension sql. Connection can be defined as follows:

```
# Connection to the database
conn = psycopg2.connect(
 dbname='database_name',
 user='user_name',
 password='user_password',
 host='host_ip',
 port='host_port'
)
# To open a cursor to perform database operations
cur = conn.cursor()
```

where instead of *database_name*, *user_name*, *user_password*, *host_ip* and *host_port* the corresponding data is required.

This allows the script to extract data from tables of the database. For example, the execution of the SQL query from the script can be done as follows:

```
cur.execute(
 "SELECT attack_actions.id, attack_actions.name, attack_actions.description "
 "FROM attackers_and_actions, attack_actions "
 "WHERE attackers_and_actions.action_id = attack_actions.id "
 "AND attackers_and_actions.access_id <= %s "
 "AND attackers_and_actions.knowledge_id <= %s "
 "AND attackers_and_actions.resources_id <= %s",
 (access_type, knowledge_type, resources_type)
)
attacker_actions = cur.fetchall()
```

And with help of the sql extension of psycopg2:

```
cur.execute(
 sql.SQL(
 "SELECT option_id FROM {} "
 "WHERE requirement_id = %s "
 "AND option_id = ANY(%s)"
 ).format(sql.Identifier(temp_table_name + '_and_compatibility')),
 (temp_id, list(temp_options),)
)
```

The main difference is the possibility to provide a database table name as a variable option. Such a possibility is actively used in the process of detailing the abstract system model.

The algorithm for the formation of requirements for microcontroller-based physical security systems, see Section 4.1, is implemented as the following function:

```
log_str, attacker_actions, devices_list, \
devices_requirements, devices_communications, \
devices_links, devices_bases = \
 self.requirements_formation(
 log_str, system_security_elements_set
 )
```

where *attacker_actions*, *devices_list*, *devices_links*, *devices_requirements*, *devices_communications* and *devices_bases* are the output data of the algorithm described in Section 4.1, while *log_str* is used to collect the work log of the algorithm and output it to the operator through the interface.

This algorithm consists of 6 stages, while the last stage consists of 7 sub-stages. Let's consider the implementation of each stage in more detail.

First stage is about initialization of data structures:

```
devices_list, devices_requirements, devices_communications, \
devices_links, devices_bases = requirements_formation_initialization()
```

Second stage is about getting attack actions possible for the attacker:

```
access_type, knowledge_type, resources_type, \
task_1, task_2, task_3 = \
 self.input_data_get()

log_str, attacker_actions = \
 attacker_actions_get(log_str, access_type, knowledge_type, resources_type)
```

Third stage is about getting security elements to prevent attack actions:

```
log_str, system_security_elements_set = \
 system_security_elements_get(
 log_str, attacker_actions, system_security_elements_set
 )
```

Fourth stage is about getting abilities of the designed system:

```
log_str, system_abilities = \
 system_abilities_get(log_str, task_1, task_2, task_3)
```

Fifth stage is about getting requirements of the designed system:

```
log_str, system_requirements = \
 system_requirements_get(log_str, system_abilities)
```

Sixth stage is about getting data of devices of the system:

```
# 6.1. Adding device to the list
devices_list.append(dev_str)

# 6.2. Getting tasks required from the device to be designed
log_str, device_tasks = \
 device_tasks_get(log_str, req_str, requirement[0])

# 6.3. Getting abilities that are connected with device tasks
log_str, device_abilities = \
 device_abilities_get(log_str, requirement[0])

# 6.4. Getting requirements that are connected with device abilities
log_str, device_requirements = \
 device_requirements_get(log_str, device_abilities)

# 6.5. Getting device base in accordance with device requirements
log_str, device_base = \
 device_base_get(log_str, device_requirements)

# 6.6. Getting types of communications that are possible for device
log_str, communications = \
 types_of_communication_get(log_str, device_base[0][0])

# 6.7. Getting links that are possible for device
links = \
 device_links_get(device_abilities)
```

The algorithm for the formation of components compositions for microcontroller-based physical security systems, see Section 4.2, is implemented as the following function:

```
log_str, system_recommendations, abstract_system_arr, abstract_links_arr = \
 self.secure_system_components_get(
 log_str, devices_list,
 devices_bases, devices_requirements, devices_links,
 devices_communications, attacker_actions,
 abstract_system_arr, abstract_links_arr,
 system_recommendations
 )
```

where *system_recommendations*, *abstract_system_arr* and *abstract_links_arr* are the output data of the algorithm described in Section 4.2, while *log_str* is used to collect the work log of the algorithm and output it to the operator.

This algorithm consists of 2 stages, while the last stage consists of 5 sub-stages. Let's consider the implementation of each stage in more detail.

First stage is about initialization of data structures:

```
log_str, system_recommendations, abstract_system_arr, abstract_links_arr = \
 self.secure_system_components_get(
 log_str, devices_list,
 devices_bases, devices_requirements, devices_links,
 devices_communications, attacker_actions,
 abstract_system_arr, abstract_links_arr,
 system_recommendations
 )
```

Second stage is about getting component composition of devices:

```
# 2.1. Getting abstract elements of device with their sub-elements
log_str, abstract_device_arr = \
 abstract_elements_with_subelements_get(
 log_str, devices_bases[device][0][0], devices_requirements[device]
 )

# 2.2. Getting attack actions that are possible for device
log_str, possible_actions_list = \
 possible_actions_get(
 log_str, devices_communications[device],
 abstract_device_arr, attacker_actions
 )

# 2.3. Getting additional abstract elements and sub-elements based on security ones
log_str, abstract_device_arr, security_elements = \
 abstract_elements_and_subelements_additional_get(
 log_str, possible_actions_list, abstract_device_arr
 )

# 2.4.1 Getting recommendations for device implementation based on security elements
log_str, device_recommendations_set = \
 device_recommendations_get(log_str, security_elements)
```

```
# 2.4.2 Getting recommendations for system implementation based on security elements
system_recommendations = \
 system_recommendations_get(security_elements, system_recommendations)

# 2.5. Saving data about device
abstract_system_arr[device] = abstract_device_arr
abstract_links_arr[device] = devices_links[device]
system_recommendations[device] = device_recommendations_set
```

The algorithm for the design of abstract models of microcontroller-based physical security systems, see Section 4.3, is implemented as the following function:

```
global abstract_system_composition

self.design_abstract_model(
 system_recommendations,
 abstract_system_arr,
 abstract_links_arr,
 system_security_elements_set
)
```

where *abstract_system_composition* is the output data of the algorithm described in Section 4.3, while the output of the algorithm is provided directly to the interface in JSON format:

```
composition_json = json.dumps(
 abstract_system_composition,
 indent=2,
 separators=(",", ": ")
)
log_array["abstract"] = composition_json
```

This algorithm consists of 7 stages, let's consider them in more detail.

First stage is about initialization of the abstract model:

```
global abstract_system_composition

abstract_system_composition["devices"] = {}
abstract_system_composition["recommendations"] = {}
abstract_system_composition["links"] = {}
```

Second stage is about generation of the system security recommendations:

```
abstract_model_system_recommendations(
       system_recommendations
)
```

Third stage is about generation of the system devices:

```
device_keys, devices_elements_keys = \
 abstract_model_devices(system_recommendations, abstract_system_arr)
```

Fourth stage is about generation of links between devices:

```
links_dep_elements = \
        abstract_model_links(
                device_keys, devices_elements_keys, abstract_links_arr
        )
```

Fifth stage is about generation of requirements for links:

```
abstract_model_links_requirements(
        links_dep_elements, system_security_elements_set
)
```

Sixth stage is about generation of dependencies between elements:

```
abstract_model_devices_dependencies_and_requirements()
```

Seventh stage is about generation of the hierarchy of elements:

```
abstract_model_devices_hierarchy()
```

The algorithm for the design of detailed models of microcontroller-based physical security systems, see Section 4.4, is implemented as the following functions:

```
global abstract_system_composition
# Formation of the selection dictionary:
# step-by-step representation of the selection process
detailed_model_selection_steps()
# Starting select process step-by-step
self.select_button_callback(scrollable_frame, "")
        button = ttk.Button(
                scrollable_frame,
                text="Confirm",
                command=lambda s_frame=scrollable_frame:
                self.select_button_callback(scrollable_frame, prev_label),
                default="normal"
        )
        button.pack(pady=5)
# Generation of the alternative of the system model
# based on the results of selection process
self.alternative_generation()
```

where *abstract_system_composition* is the input and output data of the algorithm described in Section 4.4 (it becomes output data after the detailing process), while the output of the algorithm is provided directly to the interface in JSON format:

```
composition_json = json.dumps(
 abstract_system_composition,
 indent=2,
 separators=(",", ": ")
)
log_array["detailed"] = composition_json
```

This algorithm consists of 6 stages, let's consider them in more detail.

First stage is about initialization of the data structures:

```
global select_step
global develop_dict

scrollable_frame = \
 self.selection_initialization()
```

Second stage is about generation of selection steps based on devices links:

```
global select_dict
global select_step
# First steps are always related to links between devices
selection_steps_based_on_links()
```

Third stage is about generation of selection steps based on devices components:

```
global select_dict
global select_step
# After links, steps are related to system devices, one by one
selection_steps_based_on_devices()
```

Fourth stage is about saving data of selected options:

```
select_dict[select_key]["selected"]["table"] = temp_table_name
select_dict[step_key]["selected"]["db_id"] = temp_id
```

Fifth stage is about detailing of the abstract system model:

```
# Extraction of data about selected links and elements step-by-step
temp_selected_elements = \
        abstract_model_detailing()
```

Sixth stage is about calculation of the parameters of devices:

```
# Calculation of parameters of devices based on parameters of their components
for device_key in temp_selected_elements:
        output_str = \
                device_parameters_calculation(
                        output_str, device_key, temp_selected_elements
                )
```

Thus, all algorithms from Chapter 4 are implemented in the script, while their combination is representing the base of the design methodology from Section 4.5. Note that the content of the script is described on a very high level to provide a general idea and connect it with descriptions of algorithms. For more detail, please, download its source code together with the dump of the database using the following link: https://github.com/levshun/PhD-mcbpss_design.

5.4. Interface of the software implementation

The interface is an important part of the software implementation, because it provides a possibility for the operator to work with the design methodology from Chapter 4. As was mentioned, the work process of the methodology is mostly automated, while involvement of the operator is required during transformation of wishes of stakeholder into requirements and limitations and optional at the stage of selection of the concrete implementations of elements among suitable ones during the process of detailing the abstract system model. Alternatively, the methodology can select implementations on its own. So, using the developed interface, the operator can set the parameters of the attacker, against which the system is required to be protected, as well as tasks of the system.

The interface of the application after launch is shown in Figure 101.

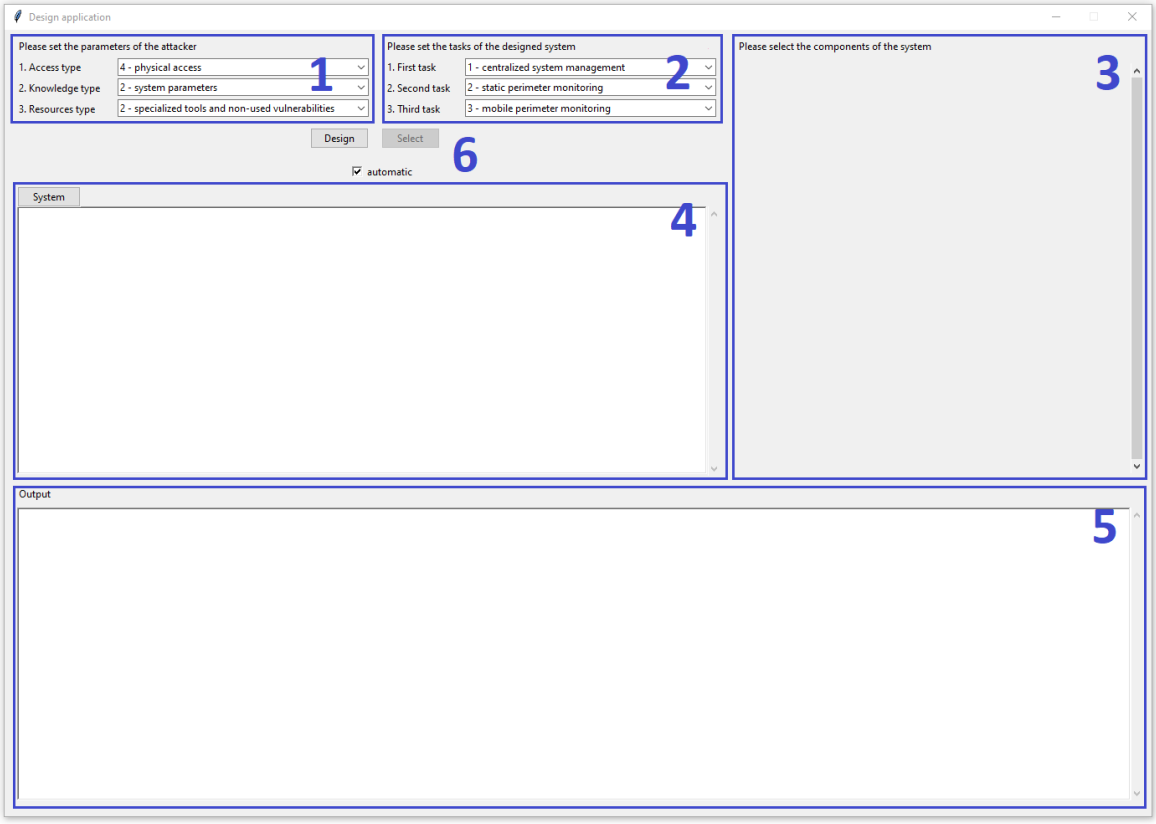

Figure 101. Interface of the application: state after launch

The interface of the application consists of **6 main parts**:

1. *Input of the parameters of the attacker* against which the designed microcontroller-based physical security system needs to be protected.
2. *Input of the tasks* that need to be solved by the designed microcontroller-based physical security system.
3. *Frame to display the process of selection* of components of the designed microcontroller-based physical security system.

4. *Frame to display the log of the work of the design methodology* for microcontroller-based physical security systems.
5. *Frame to display the results of work of the design methodology* for microcontroller-based physical security systems.
6. *Control buttons* of the application.

Let's consider each part of the interface in more detail.

Part 1. Input of the parameters of the attacker is based on the model of the attacker presented in Chapter 3 and consists of 3 parameters: access type, knowledge type and resources type, see Figure 102. Each parameter selection is based on *ttk.Combobox* and represented as drop-down lists.

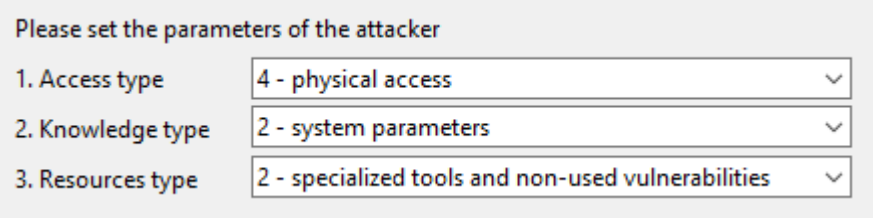

Figure 102. Interface of the application: input of parameters of the attacker

Part 2. Input of the tasks for the designed system is based on the selection of general tasks of the system, that were presented in Chapter 4. For this demo, the number of possible tasks is limited to 3, see Figure 103. Each task selection is based on *ttk.Combobox* and represented as drop-down lists.

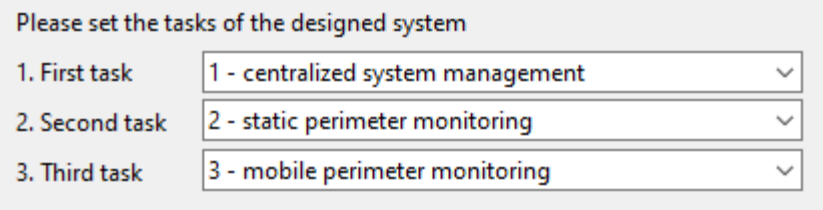

Figure 103. Interface of the application: input of tasks the system

Part 3. Frame of the components selection process displays options of communication protocols and interfaces, single-board computers, controllers and components to the operator, see Figure 104. The choice made by the operator determines the detailed model of the designed system, see Chapter 4. It is important to note that the selection process is displayed step-by-step without the possibility of changing previously made decisions. Moreover, since each choice made affects the number of options available in subsequent steps, if there is only one option for selection, the choice is made automatically.

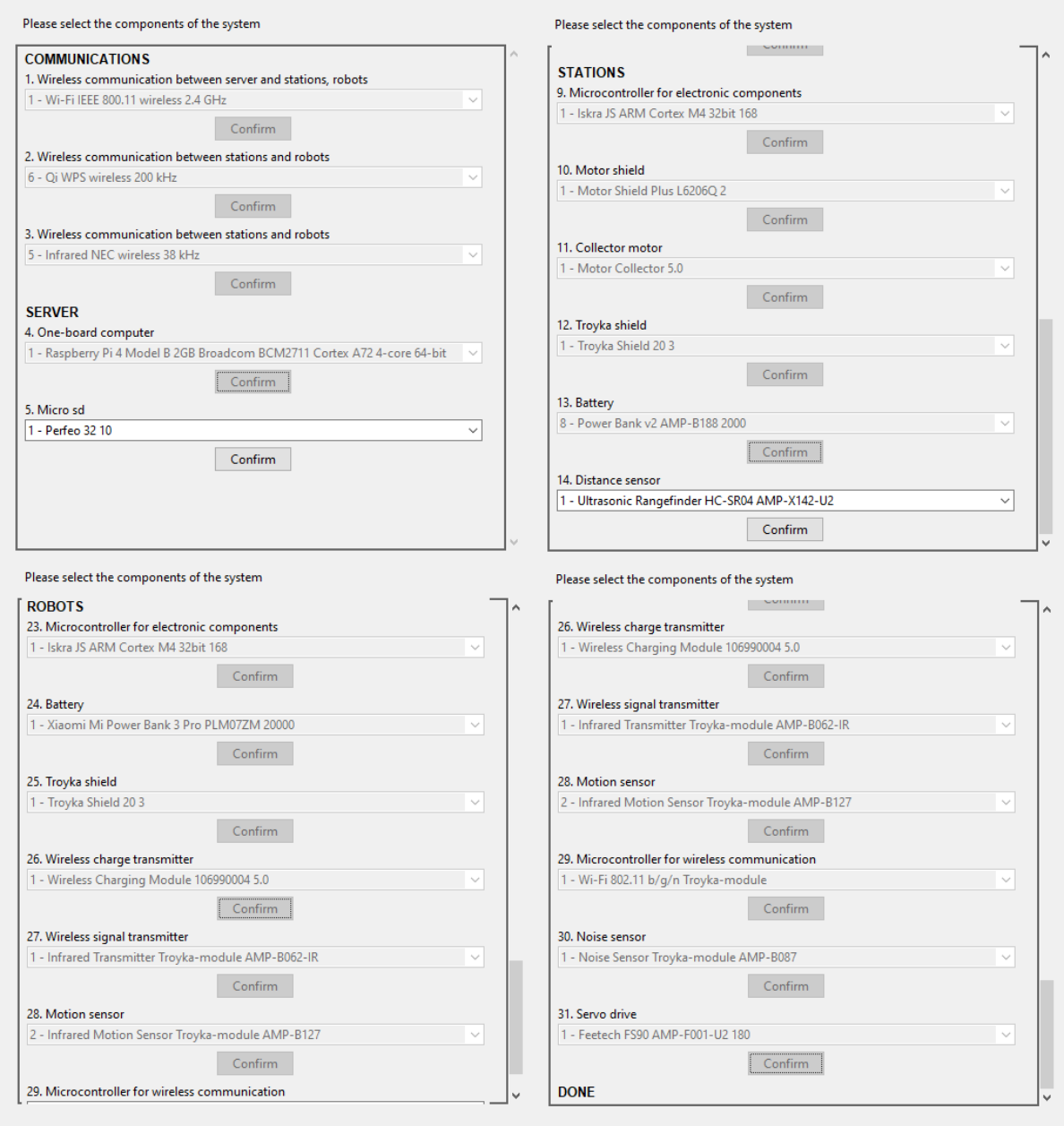

Figure 104. Interface of the application: frame of the components selection process

The interface for each selection step is represented as a drop-down list with options and based on the combination of *ttk.Combobox*, *ttk.Button* and *ttk.Label*. The selection process begins with choice of communications between devices of the designed system and after that continues with components of each device, namely server, stations and robots, are selected, see Chapter 4.

Part 4. Frame for the design methodology work process log displays separate logs for the designed system, its devices, abstract and detailed models, see Figure 105. Display of log is based on *tk.Text* and *tk.Scrollbar* objects that are linked together.

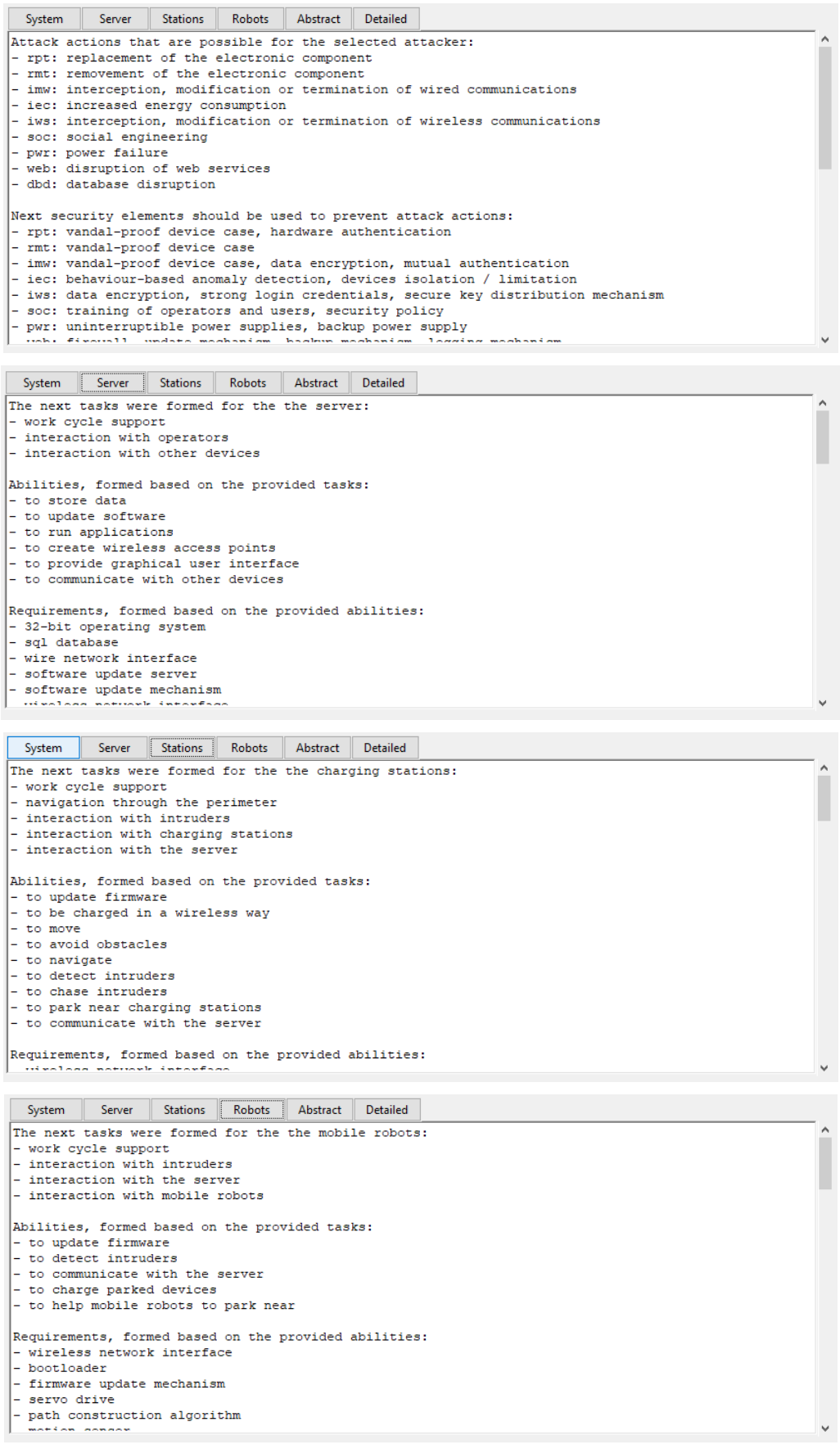

Figure 105. Interface of the application: frame for the methodology work process log

System log contains information about attack actions that are possible for the selected attacker, security elements that should be used to prevent them, system abilities that were formed based on provided tasks, requirements that were formed based on these abilities and recommendations for the system implementation. For more detail, see Chapter 4.

Server, *Stations* and *Robots logs* contain information about tasks that were formed for each device, abilities that were formed based on this tasks, requirements that were formed based on this abilities, base of this device, its abstract elements, sub-elements and types of communication, attack actions that are possible based on types of communication, abstract elements and attacker parameters, security elements to prevent attack actions, additional elements of the device, additional sub-elements of the device, generated set of device components and recommendations for the server implementation. For more detail, see Chapter 4.

Abstract log contains the abstract system model in JSON format, see Figure 106:

- system contains devices, recommendations and links;
- each device contains id, name, components and recommendations;
- each recommendation (system/device) contains id and name;
- components of each device contain elements, each of which has its own id, name, components, links, requirements, dependencies and parent tag;
- each element link contains id, type and parties;
- each element requirement contains id and name;
- each element dependency contains keys of elements that depend on them;
- element parent tag contains the key of the element that contains this element as its sub-element.

Figure 106. Abstract system model in JSON format

The abstract model of the system is a mapping of the extendable set-based hierarchical relational model from Chapter 3. For more detail, see Chapter 4.

Detailed log contains the detailed system model in JSON format, see Figure 107. Detailed model of the system is an extended version of the abstract model. For more detail, see Chapter 4.

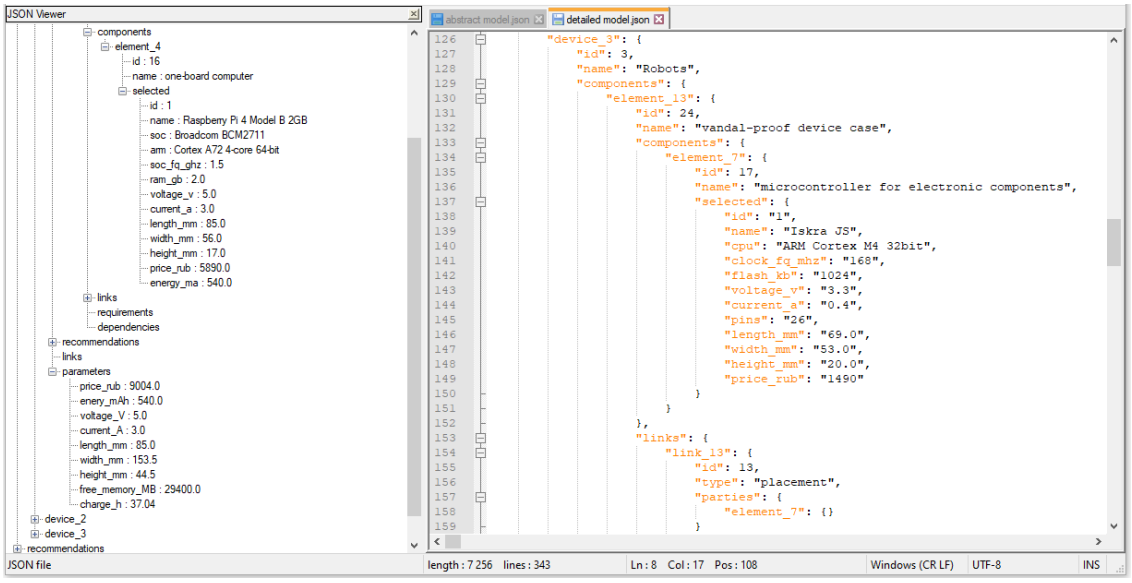

Figure 107. Detailed system model in JSON format

Part 5. Frame for the results of the design methodology displays for each device the list of its components that were selected with their parameters as well as the list of components that are required to be developed or configured with required algorithms or settings, see Figure 108. In addition, parameters of each device as well as security recommendations for their implementation are displayed.

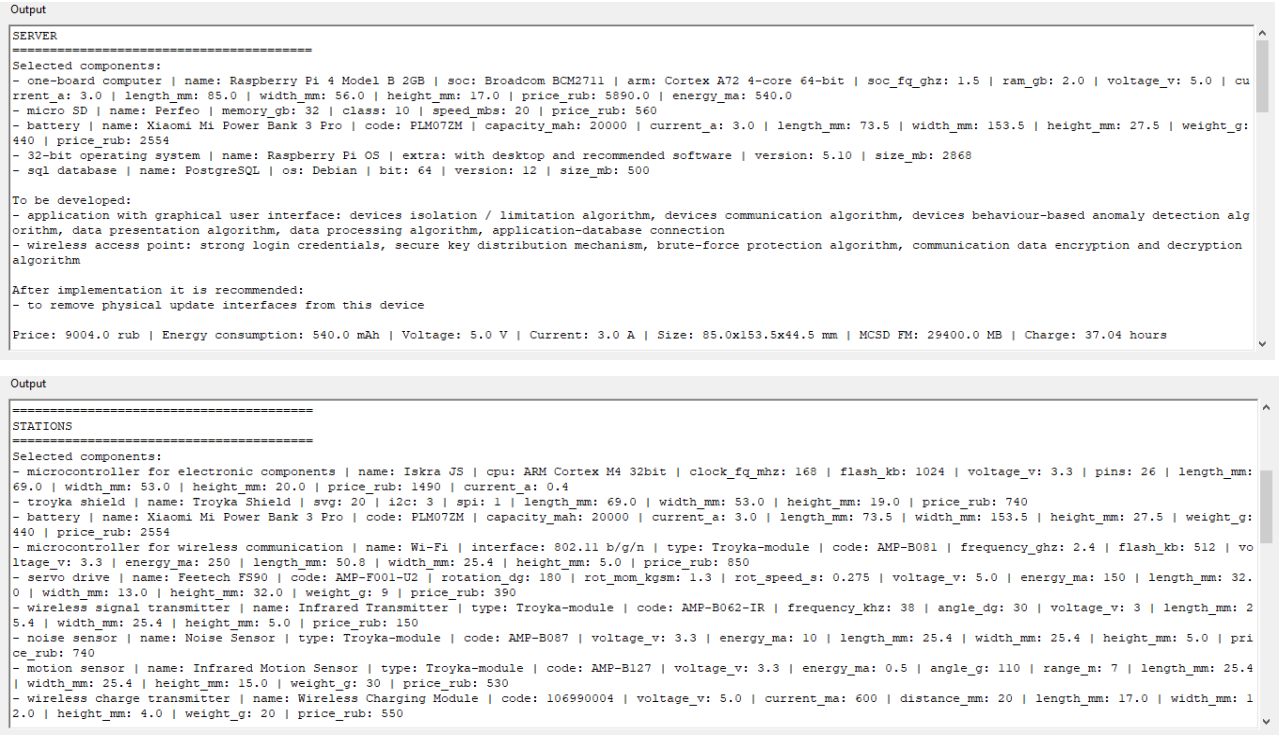

Figure 108. Interface of the application: for the results of the design methodology

Display of the results is based on *tk.Text* and *tk.Scrollbar* that are linked together.

Part 6. Control buttons of the application are represented as *Design* and *Select* buttons (*ttk.Button*) and *automatic* checkbutton (*tk.Checkbutton*), see Figure 109.

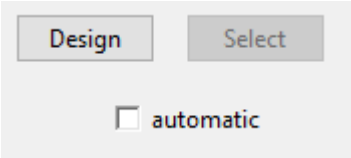

Figure 109. Interface of the application: control buttons

Design button starts the design process for the abstract model of the microcontroller-based physical security system. *Select button* starts the selection process for the elements of the system to design its detailed model. *Automatic checkbutton* switches from manual selection process (by the operator) to automated (by the methodology).

5.5. Conclusions on Chapter 5

The architecture of the software implementation of the methodology consists of the Python script, PostgreSQL database and Tkinter interface. PostgreSQL database is required to store data about the extendable set-based hierarchical relational model of microcontroller-based physical security systems from Chapter 3, as well as data for algorithms and methodology from Chapter 4. Python script represents the implementation of the algorithms and methodology from Chapter 4. Each algorithm is implemented as a number of functions, while all functions are connected with each other in a single methodology. Tkinter interface is required to receive input data from the operator, namely, parameters of the attacker and tasks of the designed system, as well as to provide the output data to him or her.

The developed database contains more than 100 tables, while the database initialization contains more than 2300 lines of PL/pgSQL queries. The structure of the database is huge, that is why it was decided to divide its description on the following parts of storage: (1) attacker, attack actions and security elements; (2) tasks, abilities and requirements; (3) abstract elements, sub-elements and links; and (4) detailed elements. The structure of the first part contains 11 tables, while the structure of the second — 11, third — 27 and fourth — more than 50. Moreover, it was decided to divide the third one into abstract sub-parts, namely, (3.1) elements, (3.2) sub-elements, (3.3) links and (3.4) recommendations. The description of the database contains information about connections between its parts, structure of its tables, connections between tables and examples of their content as well as examples of SQL queries to extract data from the database.

The developed script contains more than 3000 lines of code and works with such imports as psycopg2, tkinter, pygubu, networkx, json, functools and time. The role of the script is to implement algorithms from Chapter 4, combine them together into the design methodology from Section 4.5 and provide connections between the database from Section 5.2 and the interface from Section 5.4. The script connects itself with the developed interface with the help of the pygubu library. This allows the script to get access to objects of the interface and control them: default state, selected values, callback functions and links between them can be defined. The script connects itself with the developed database with the help of the psycopg2 library and its extension sql. This allows the script to extract data from tables of the database. The description of the script contains the information about the implementation of each algorithm from Chapter 4 with all their stages.

The interface is an important part of the software implementation, because it provides a possibility for the operator to work with the design methodology from Chapter 4. As was mentioned, the work process of the methodology is mostly automated, while involvement of the operator is required during transformation of wishes of stakeholder into requirements and limitations and optional at the stage of selection of the concrete implementations of elements among suitable ones during the process of detailing the abstract system model. Alternatively, the methodology can select implementations on its own.

The interface of the application consists of 6 main parts: (1) input of the parameters of the attacker against which the designed system needs to be protected; (2) input of the tasks that need to be solved by the designed system; (3) frame to display the selection process of the designed system components; (4) frame to display the work log of the design methodology; (5) frame to display the results of the design methodology and (6) control buttons of the application.

The source code of the script, the dump of the database as well as the file of graphical user interface are available for download using the following link: https://github.com/levshun/PhD-mcbpss_design.

The experimental evaluation of the software implementation of the methodology for the design of microcontroller-based physical security systems is presented in the next chapter. It is evaluated in accordance with requirements from Section 1.4 and problem statement from Section 1.5 based on methods from Chapter 2.

Chapter 6. Experimental evaluation of the methodology for the design of microcontroller-based physical security systems

This chapter describes the evaluation of the methodology for the design of microcontroller-based physical security systems. It contains the description of the experiment of its application to the use case of mobile robots for the perimeter monitoring as well as results of the evaluation of its software implementation.

6.1. Experiment description

As was mentioned in Chapter 5, software implementation of the design methodology is an application that consists of the Python script, PostgreSQL database and Tkinter interface. For the experiment, it was decided to run this application on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM.

In accordance with Sections 1.4 and 1.5 it is required to evaluate the software implementation based on functional and non-functional requirements as well as compare it with commercial and scientific solutions.

Functional requirements are representing the list of functions that are defining the actions that the software implementation must perform:

1. Building an abstract representation of the designed system.
2. Finding a trade-off between the resources spent and security.
3. No restrictions on platforms and architectures of the devices to be designed.
4. The extensibility of the design process.
5. Taking into account the physical layer of designed systems.

Non-functional requirements are describing the system requirements and constraints imposed on the resources consumed by the software implementation:

1. Time required for the design process of the abstract model of the system should be less than 1 seconds.
2. Time required for the design process of the detailed model of the system should be less than 4 seconds.
3. Number of resources required for the design process of the system should be less than 25% of the computer resources.
4. Number of levels of the system, the security of which can be ensured, should be maximized.
5. Number of classes of attack actions against which the system can be protected should be maximized.

First and second non-functional requirements are representing the *time consumption* requirement and checked in accordance with the method from Section 2.1.

Such requirements for time consumption were chosen for the design process of the microcontroller-based physical security system that contains 3 types of devices, while each type of devices consists of not less than 5 elements with sub-elements, second type — not less than 10 elements with sub-elements and third type — not less than 15 elements with sub-elements. It is important to note that the designed system should also take into account links between devices and their elements, security recommendations to their implementation, requirements for links and elements as well as dependencies between them.

In addition, it was decided to investigate dependencies between the design time and parameters of the attacker, against which the system is required to be protected.

Third non-functional requirement is representing the *resource consumption* requirement and checked in accordance with the method from Section 2.2.

Fourth and fifth non-functional requirements are representing the *validity* requirement and checked in accordance with the method from Section 2.3.

Comparison with commercial solutions is required in terms of levels of the system, the security of which can be ensured:

1. Controllers, components and their communications.
2. Controllers and their communications inside devices.
3. Devices and their communications with each other.
4. Systems and their communications with each other.

Comparison with scientific solutions is required in terms of classes of attack actions, against which the system can be protected:

1. Components and their communications with controllers.
2. Controllers and their communications with other controllers.
3. Devices and their communications with other devices.
4. System and its communications with other systems.

Summarizing the above, the plan of the experiment is as follows:

1. Description of the system to be designed.
2. Manual fulfillment of the database with data about the system.
3. Analysis of the application compliance with functional requirements.
4. Analysis of time consumption of the application.
5. Analysis of resource consumption of the application.
6. Analysis of the validity of the application.
7. Comparison of the application with scientific solutions.
8. Comparison of the application with commercial solutions.
9. Investigation of the dependencies between the design time and attacker.

6.1.1. Description of the system

For the experiment, it was decided to design a microcontroller-based physical security system that provides perimeter monitoring based on mobile robots, see Figure 110. This system contains a server as well as multiple mobile robots and charging stations with different controllers and components. Robots (mobile objects) and stations (static objects) are monitoring the perimeter via different sensors based on the server instructions. And if the battery of one of the robots is low, it moves to the nearest free charging station. The information about the perimeter map, locations of robots and stations as well as the charge state of robots and occupancy of stations is stored on the server.

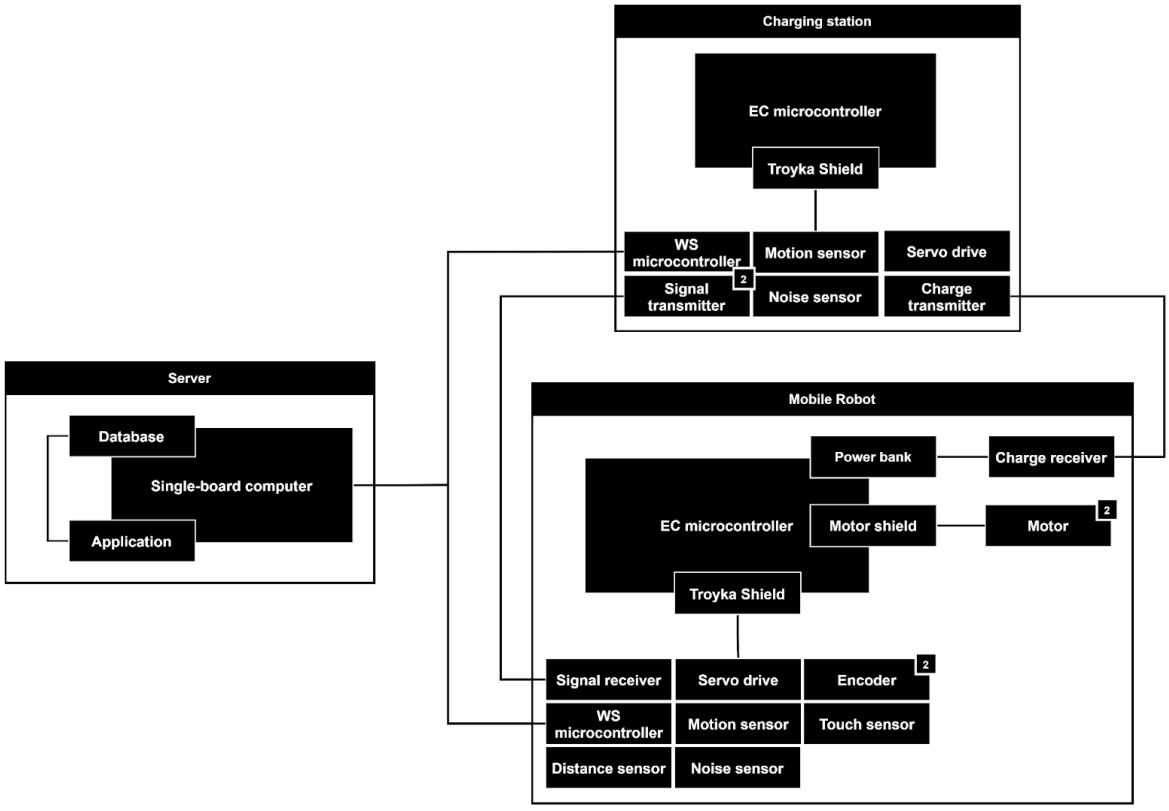

Figure 110. Architecture of the perimeter monitoring system

Such a system was chosen due to the presence of several types of devices, multiple communications between them, as well as the need to use many different elements for each device in the system (server consists of 8 elements with sub-elements, station — 12 and robot — 17, which means that such a system is appropriate in accordance with the provided requirements). Moreover, there are links between devices of the system and elements of devices, requirements for links and elements as well as dependencies between them. Also, during the design of such a system it is necessary to ensure not only its functionality (perimeter monitoring), but also to ensure that the system is secure against attacks on it.

6.1.2. Tasks, abilities and requirements of the system and its devices

As was mentioned in Section 4.1, extraction of requirements for the designed system starts from its tasks that are formulated by the operator in accordance with wishes of the stakeholder. Tasks of the microcontroller-based physical security system, described in Section 6.1.1, can be represented as follows:

- centralized system management;
- static perimeter monitoring;
- mobile perimeter monitoring;
- appropriate level of security.

Note that an appropriate level of security is set according to parameters of the attacker model presented in Section 3.2 — access, knowledge and resources types. Links between these tasks and abilities as well as requirements are considered in more detail in Table 11.

Table 11. Tasks, abilities and requirements of the designed system

Task	Ability	Requirement	Dependency
centralized system management	to store and process system data	device that represents *the server* of the system	
	to run executable applications		
	to download and install software updates		
	to create wireless access points		
	to communicate with mobile robots		
	to communicate with charging stations		
	to provide user interface for operators of the system		
static perimeter monitoring	to provide wireless charging	devices that represent *charging stations* of the system	to provide *static perimeter monitoring*, the task of *centralized system management* should already be satisfied
	to monitor the perimeter nearby		
	to communicate with mobile robots		
	to communicate with the server of the system		

Task	Ability	Requirement	Dependency
mobile perimeter monitoring	to be charged wirelessly	devices that represent *mobile robots* of the system	to provide *mobile perimeter monitoring*, the tasks of *centralized system management* and *static perimeter monitoring* should already be satisfied
	to navigate through the perimeter		
	to detect and chase intruders		
	to communicate with charging stations		
	to communicate with the server of the system		
appropriate level of security	to be secure against attackers with $ac = 4$, $kn = 2, rs = 2$	security requirements should be taken into account during formation of all devices of the system	

It is required to store tasks of the system in the *system_tasks* table of the database, abilities — *system_abilities*, requirements — *system_requirements*, while connections between them in such tables as *system_tasks_and_abilities* and *system_abilities_and_requirements*. For more detail, see Section 5.2.2.

As noted in Table 11, requirements can be divided into requirements for the server, mobile robots and charging stations of the system as well as security requirements. Note that security requirements should be taken into account not only on the system level, but also during the design of all its devices.

Let's consider requirements for *the server of the system* in more detail.

Once again it is required to analyze tasks for such a device, connect them with abilities and requirements, see Table 12. Tasks of the server can be divided into:

- work cycle support;
- interaction with the operator;
- interaction with other devices;
- appropriate level of security.

Note that on the level of devices it becomes possible to connect requirements with controllers and different components as well as parts of the software and firmware.

In addition, requirements for the designed system must be connected with corresponding tasks of its devices. It is required to store those connections in the *system_requirements_and_tasks* table of the database, see Section 5.2.2.

Table 12. Tasks, abilities and requirements related to the server

Task	Ability	Requirement	Dependency
work cycle support	to store data	32-bit operating system	
		sql database	
	to update software	wire network interface	
		software update server	
		software update mechanism	
	to run applications	32-bit operating system	
	to create wireless access points	32-bit operating system	
		wireless network interface	
		access points configuration mechanism	
interaction with operators	to provide graphical user interface	application with GUI	to provide *interaction with operators*, the task of *work cycle support* should already be satisfied
		app-db connection	
		data processing algorithm	
		data presentation algorithm	
interaction with other devices	to communicate with other devices	wireless network interface	to provide *interaction with other devices*, the task of *work cycle support* should already be satisfied
		devices communication algorithm	
appropriate level of security	to be secure against attackers with $ac = 4$, $kn = 2$, $rs = 2$	security should be taken into account during the formation of all elements of the device	

It is required to store tasks of the device in the *device_tasks* table of the database, abilities — *device_abilities*, requirements — *device_requirements*, while connections between them in *device_tasks_and_abilities* and *device_abilities_and_requirements*. For more detail, see Section 5.2.2.

The following device, whose requirements must be considered in more detail, is *one of the charging stations of the system*. Its tasks can be divided into:

- work cycle support;
- interaction with intruders;
- interaction with mobile robots;
- interaction with the server;
- appropriate level of security.

Links between tasks, abilities and requirements of one of the charging stations are considered in more detail in Table 13.

Table 13. Tasks, abilities and requirements related to the charging stations

Task	Ability	Requirement	Dependency
work cycle support	to update firmware	wireless network interface	
		bootloader	
		firmware update mechanism	
	to charge parked devices	wireless charge transmitter	
interaction with intruders	to detect intruders	motion sensor	to provide *interaction with intruders*, the task of *work cycle support* should already be satisfied
		noise sensor	
		servo drive	
		intruder detection algorithm	
interaction with parking devices	to help mobile devices to park near	wireless signal transmitter	to provide *interaction with parking devices*, the task of *work cycle support* should already be satisfied
		parking direction algorithm	
interaction with the server	to communicate with the server	wireless network interface	to provide *interaction with the server*, the task of *work cycle support* should already be satisfied
		server communication algorithm	
appropriate level of security	to be secure against attackers with $ac = 4$, $kn = 2, rs = 2$	security should be taken into account during the formation of all elements of the device	

Once again, it is required to store this data in the database: tasks — *device_tasks*, abilities — *device_abilities*, requirements — *device_requirements*, while connections between them in *device_tasks_and_abilities* and *device_abilities_and_requirements*. For more detail, see Section 5.2.2.

The last device, which requirements are required to be considered in more detail, is *one of the mobile robots of the system*. Its tasks can be divided into:

- work cycle support;
- perimeter monitoring;
- interaction with intruders;
- interaction with charging stations;
- interaction with the server;
- appropriate level of security.

Links between tasks, abilities and requirements of one of the mobile robots are considered in more detail in Table 14.

Table 14. Tasks, abilities and requirements related to the mobile robots

Task	Ability	Requirement	Dependency
work cycle support	to update firmware	wireless network interface	
		bootloader	
		firmware update mechanism	
	to be charged in a wireless way	wireless charge receiver	*battery* should provide power supply for *8 hours*
		battery	
		charge monitoring algorithm	
perimeter monitoring	to move	collector motor	to *move*, the *work cycle support* task should already be satisfied
		movement algorithm	
	to avoid obstacles	distance sensor	to *avoid obstacles*, each robot should already have an ability to *move*
		touch sensor	
		servo drive	
		obstacles detection algorithm	
		obstacles avoidance algorithm	
	to navigate	encoder	to *navigate*, each robot should already have and ability to *avoid obstacles*
		map construction algorithm	
		path construction algorithm	
interaction with intruders	to detect intruders	motion sensor	to *detect an intruder*, the *perimeter monitoring* task should already be satisfied
		noise sensor	
		servo drive	
		intruders detection algorithm	
	to chase intruders	distance sensor	to *chase intruders*, each mobile robot should already have an ability to *detect intruders*
		intruders chase algorithm	
interaction with charging stations	to park near charging stations	wireless signal receiver	to *park near charging stations*, the *perimeter monitoring* task should already be satisfied
		parking algorithm	

Task	Ability	Requirement	Dependency
interaction with the server	to communicate with the server	wireless network interface	to *communicate with the server*, the *work cycle support* task should already be satisfied
		server communication algorithm	
appropriate level of security	to be secure against attackers with $ac = 4$, $kn = 2$, $rs = 2$	security requirements should be taken into account during the formation of all elements of the server	

Once again, it is required to store this data in the database: tasks — *device_tasks*, abilities — *device_abilities*, requirements — *device_requirements*, while connections between them in *device_tasks_and_abilities* and *device_abilities_and_requirements*. For more detail, see Section 5.2.2.

It is important to note that devices of the designed system have requirements that introducing dependencies between their elements after selection:

- *wireless network interface* requirement must be satisfied for the server, charging stations and mobile robots in such a way that they can communicate with each other (selected implementations must be compatible);
- *wireless charge transmitters* of charging stations must be compatible with *wireless charge receivers* of mobile robots;
- *wireless signal transmitters* of charging stations must be compatible with *wireless signal receivers* of mobile robots.

Also, dependencies between tasks and abilities can be hierarchical, for example, such dependencies for one of the mobile robots are presented in Figure 111.

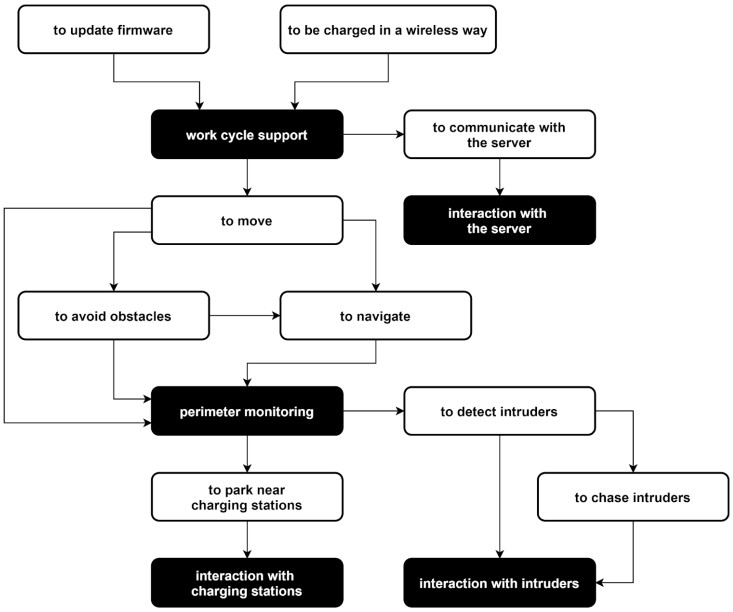

Figure 111. The hierarchy of tasks and abilities of one of the mobile robots

6.1.3. Component composition of devices of the system

As was mentioned in Section 6.1.2, the microcontroller-based physical security system from Section 6.1.1 contains three types of devices — the server, mobile robots and charging stations, while each type has its own requirements. This section is describing how the database of the software implementation, the structure of which was presented in Section 5.2, can be filled, so the algorithms from Section 4.1 and 4.2 will be able to prepare data for the abstract system representation.

It is important to note that in this section it is assumed that the database is filled with data about attackers, attack actions and security elements as well as connections between them in accordance with Chapter 3, while the structure of the database for them was presented in Section 5.2.1.

In Section 5.2.3 it was stated that in this work there are three types of bases that are possible for microcontroller-based devices, namely, such a device can be based on a *single-board computer*, *connected microcontrollers* or *microcontroller*. Moreover, the possibilities of bases are hierarchical: any single-board computer can do anything that is possible for connected microcontrollers, while having additional possibilities, any connected microcontrollers > any microcontroller and so on. Possible bases of devices must be stored in the *device_base* table of the database, while hierarchy of bases is represented by their ids in this table: more possibilities → lower id.

The process of understanding which base is required for the designed devices is based on checking its requirements. It means that each requirement for any device from Section 6.1.2 that are stored in *device_requirements* must be connected with an appropriate base from *device_base*, while appropriate means base with minimal possibilities that can be used to satisfy the requirement. For example, the requirement “32-bit operating system” requires the “single-board computer” base to be satisfied, while “wireless signal transmitter” can be satisfied by “microcontroller”. Such connections must be stored in *device_requirements_and_base*.

In accordance with the architecture of microcontroller-based physical security systems that is used in this work, devices of such systems can communicate only on four levels: *controller* ↔ *component*, *controller* ↔ *controller*, *device* ↔ *device* and *system* ↔ *system* (note: might not be all of them). Communication levels are stored in *communication_levels*, while their availability for designed devices is defined in accordance with their bases. For example, “microcontroller” base can communicate only on “controller ↔ component” level, while “connected microcontrollers” — “controller ↔ component”, “controller ↔ controller” and “device ↔ device”. It is required to store such connections in *base_and_communication*.

Possible abstract links between devices of the designed system must be stored in the *abstract_links* table of the database. For the system provided in Section 6.1.1 it

is enough to store “wireless” here. After that, stored links must be connected with abilities of devices from *device_abilities* to define which abilities are required to have one of the “wireless” links between devices. Such connections must be stored in *abstract_links_and_abilities*. For example, a “wireless” link robots ↔ stations can be extracted in accordance with “to help mobile robots to park near” ability of charging stations and “to park near charging stations” ability of mobile robots.

It is also required to connect requirements for devices with abstract elements and sub-elements that can be used to satisfy them. It is important to note that not all requirements can be represented as abstract elements. For example, the requirement “32-bit operating system” can be linked with the abstract element “32-bit operating system”, while the requirement “wire network interface” can only be used as one of requirements to the implementation of the controller of the designed device. Moreover, the requirement “charge monitoring algorithm” can be linked only with the abstract sub-element “charge monitoring algorithm”. Abstract elements must be stored in *abstract_elements*, sub-elements — *abstract_subelements*.

After that, based on the algorithm from Section 4.2, it is required to connect abstract elements with bases of designed devices, requirements for them, sub-elements, each other and security elements. Let’s consider each connection in more detail.

Connections between abstract elements and bases are stored in the *abstract_elements_and_base* table of the database. For example, base “connected microcontrollers” can be connected with “microcontroller for electronic components” and “microcontroller for wireless communication” abstract elements.

Connections between abstract elements and requirements for designed devices are stored in *abstract_elements_and_requirements*. For example, the requirement “distance sensor” can be connected with the “distance sensor” abstract element. Note that it is not obligatory for the requirement to have the same name as the corresponding abstract element. For example, this requirement can be rewritten as “to have a sensor that is able to measure the distance to the nearest obstacle”. Developed algorithms are not working with text values of requirements and interested only in their identificators in the database.

There are also inner connections between abstract elements. Such connections are required, because not all abstract elements that are necessary for the designed device can be extracted in accordance with bases and requirements. Some of them can be extracted only based on other abstract elements. For example, the abstract element “single-board computer” is connected with the “micro-SD” abstract element. Such connections must be stored in *abstract_elements_and_elements*.

Abstract sub-elements can be extracted in accordance with device requirements, but they are also dependent on abstract elements, because they represent their parts.

For example, the requirement “obstacles detection algorithm” states that the abstract element “firmware for electronic components” must have such abstract sub-elements as “obstacles detection algorithm” and “obstacles avoidance algorithm”. Such connections must be stored in *abstract_elements_and_subelements*.

In addition, abstract sub-elements can be extracted in accordance with device bases and abstract elements they are dependent on. For example, the “microcontroller” base adds the “electronic components interaction algorithm” abstract sub-element to the “firmware for electronic components” abstract element.

Connections between abstract and security elements are stored in *abstract_elements_and_security*. For example, the security element “backup power supply” can be connected with the abstract element “battery”.

Connections between abstract sub-elements and security elements are stored in *abstract_subelements_and_security*. Once again, sub-elements are representing parts of elements, that is why such connections are also depending on them. For example, the security element “data encryption” adds the abstract sub-element “communication data encryption and decryption algorithm” to the abstract elements “firmware for electronic components”.

Note that not all security elements can be interpreted as abstract elements and sub-elements. That is why in this work some of them are interpreted as recommendations to the implementation. Such recommendations are divided into one related to the designed system and the other one related to its devices.

Connections between recommendations to the implementation of devices and security elements are stored in the *device_recommendations_and_security* table of the database. For example, the security element “hidden placement of sensors” can be connected with the requirement “to hide monitoring sensors of this device”.

Connections between recommendations to the implementation of the system and security elements are stored in the *system_recommendations_and_security* table of the database. For example, the security element “training of operators and users” can be connected with the requirement “to educate operators and users of the system about social engineering attacks”.

To not describe each insert to the developed database, its dump that was used for the design of the microcontroller-based physical security system, namely, perimeter monitoring system based on mobile robots, is available for download using the following link: https://github.com/levshun/PhD-mcbpss_design.

6.2. Application of the design methodology

The application of the design methodology presented in Section 4.5 to the system presented in Section 6.1.1 can be divided into the design of its abstract and detailed models. Wherein the detailed model is an extension of the abstract one as well as the abstract model is a representation of the extendable set-based hierarchical relational model presented in Chapter 3. Let's consider the fulfillment of the database for the design of each model in more detail.

6.2.1. Abstract model of the system

The algorithm for the design of abstract models of microcontroller-based physical security systems is presented in Section 4.3. This algorithm represents the system as an abstract hierarchical model that takes into account connections between system devices, their elemental composition, dependencies between device elements and requirements for them. Let's consider database tables that must be filled for the correct work of the algorithm in more detail.

As was mentioned in Section 5.2.1, in this work abstract elements are divided into several types that are stored in *elements_types*. Connections between abstract elements and their types are stored in *abstract_elements_and_types*. For example, such abstract elements as "distance sensor" and "touch sensor" can be connected with the type "environment sensors", while "motion sensor" and "noise sensor" — "monitoring sensors". This table can also be used to calculate the number of abstract elements of a certain type.

During the formation of requirements to controllers of microcontroller-based devices, it is important to assume the number of digital and analog pins that are required to connect all necessary abstract elements as well as the amount of flash memory that is required to run the firmware with all necessary abstract sub-elements.

Connections between abstract elements and the number of pins that are required for their connection to controllers are stored in *abstract_elements_and_pins*. For example, the abstract element "motor shield" requires 4 pins of a controller, while "motion sensor" requires only 1. Based on the designed device component composition and the content of *abstract_elements_and_pins* it becomes possible to form requirements for controllers of devices that are defining the minimal number of pins that they must have.

Connections between abstract sub-elements and the amount of flash memory that is required for them are stored in *abstract_subelements_and_flash_memory*. For example, "communication data encryption and decryption algorithm" required 100 KB of flash memory, while "obstacles detection algorithm" required only 20 KB. Based on algorithms of firmware of controllers of designed devices and the content of

abstract_subelements_and_flash_memory it becomes possible to form requirements for controllers that defines the minimal amount of flash memory that they must have.

It is important to take into account dependencies between abstract elements and links during the design of the abstract system model. Such dependencies are taken into account during the selection of the implementations of elements and links during the abstract model detailing process. In this work, dependencies are divided into two types: abstract elements that are depending on abstract links and abstract elements that are depending on other abstract elements. Let's consider each type in detail.

Dependencies between abstract links and abstract elements are stored in the *links_and_dependencies* table of the database. For example, the link that is based on such abilities as "to be charged in a wireless way" and "to charge parked devices" can be connected with the following abstract elements: "wireless charge receiver", "wireless charge transmitter" and "battery". It means that after the implementation of the wireless charging link between devices is selected, the number of options for the selection of mentioned abstract elements is limited for compatibility.

Dependencies between abstract elements are stored in the following table of the database: *abstract_elements_and_dependencies*. For example, the following abstract elements are depending on the "microcontroller for electronic components" abstract element: "battery", "motor shield" and "troyka shield". It means that after the implementation of the controller that is used to work with components of the device is selected, the number of options for the selection of mentioned abstract elements is also limited for compatibility.

In this work, links are taken into account not only on the level of communications between devices of the designed system, but also on the level of communications between elements of devices. The information about the link between two elements after their combination must be stored in *abstract_links_and_elements*. For example, the combination of the following abstract elements can be connected with the abstract link "VG": "microcontroller for electronic components" and "battery". In the mentioned abstract link, "V" means voltage, while "G" means ground. Such a link represents a two-wire connection, where two elements are sharing the power supply. For more information about possible values of abstract links, see Section 3.1.

The information about the possibility to combine one abstract element with another must be stored in *abstract_elements_combination*. For example, the abstract element "32-bit operating system" can be connected with the following elements: "sql database", "application with graphical user interface" and "wireless access point". It means that each of mentioned abstract elements can be a part of "32-bit operating system": "sql database" and "application with graphical user interface" can be installed and executed with it help, while "wireless access point" can be configured based on the operating system functionality.

6.2.2. Detailed model of the system

The algorithm for the design of detailed models of microcontroller-based physical security systems is presented in Section 4.4. Detailed model preserves and expands the structure of the abstract one and takes into account compatibility, requirements, dependencies and hierarchy of system elements. Let's consider database tables that must be filled for the correct work of the algorithm in more detail.

The process of transition from abstract to detailed models is a step-by-step process. Each step represents the selection of implementations of one of the abstract elements, while the sequence of steps is formed in accordance with their hierarchy and dependencies. Moreover, after each step the number of options for further steps is limited in accordance with compatibility. Let's consider database tables that must be filled for the correct work of the algorithm in more detail.

Implementations of links between devices of the designed system must be stored in the *links_between_devices* table of the database. This table describes if the implementation of the link is wireless, directed; transfers data, charge or signals; involves the creation of access points; provides encryption, authentication. Moreover, it describes its speed and range. All this information is used to check if the implementation satisfies requirements that were formed during the design of the abstract system model. For example, Wi-Fi communication:

```
{
 "name": "Wi-Fi",
 "interface": "IEEE 800.11",
 "protocol": "wireless 2.4 GHz",
 "wireless": "true",
 "directed": "false",
 "data": "true",
 "charge": "false",
 "signal": "false",
 "access_point": "true",
 "encryption": "true",
 "authentication": "true",
 "range": 40,
 "speed": 20
}
```

It means that Wi-Fi communication is wireless and transfers data on 2.4 GHz frequency using IEEE 800.11 protocol. Moreover, it provides a possibility to create access points, has encryption and authentication (for example, WPA2-PSK). Its range is near 40 meters, while speed is around 20 Mbps (such a value is taken because microcontroller-based devices are not as powerful as modern routers). This description also states that the Wi-Fi communication is not directed and can't be used for charging as well as for low-level communications like signal transferring.

The process of detailing mobile robots of the system is presented in Figure 112.

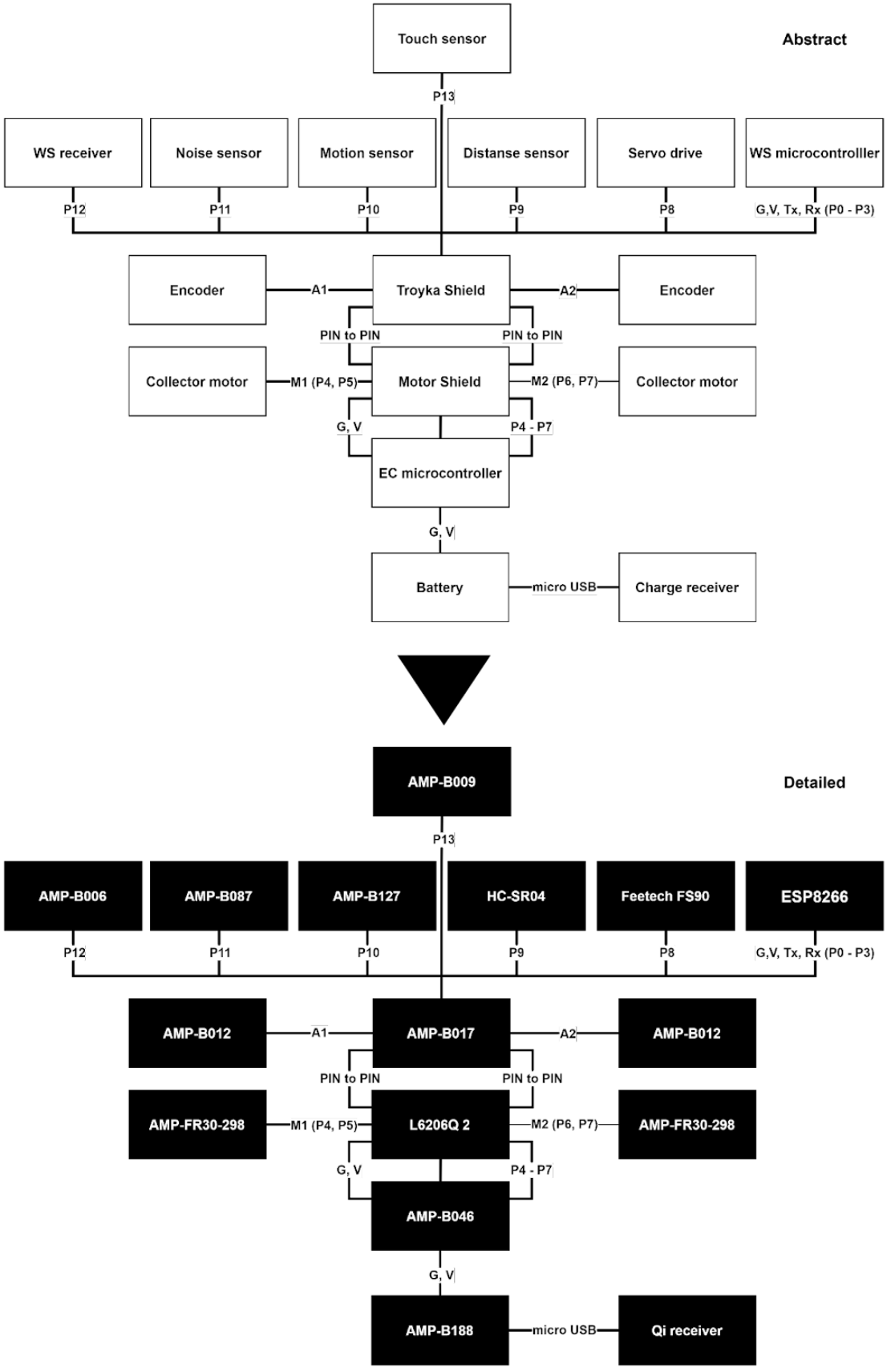

Figure 112. Detailing of one of the mobile robots of the designed system

It means that each abstract element of each device of the system must be detailed based on the selection of its implementation that satisfies given requirements, while the number of elements to be detailed differs from device-to-device. It means that the algorithm requires an abstract way to work with any number of abstract elements and extract necessary knowledge about their implementations from the database.

Such work of the algorithm is possible because of the following table of the database: *selectable_elements_and_db_tables*. This table connects abstract elements with names of database tables, where their implementations are stored. For example, implementations of the abstract element “32-bit operating system” are stored in the *operating_systems_32bit* table, while “microcontroller for electronic components” — *microcontrollers_for_electronic_components*. It means that the algorithm can provide the database id of the abstract element to find out in which table its implementations are stored.

After all possible implementations of the abstract element are known for the algorithm, it is required to check them in terms of requirements, compatibility and dependencies. Let’s consider what tables of the database are used for this.

Requirements are checked based on the content of the database table, extracted from *selectable_elements_and_db_tables*. Each requirement from the abstract system model has the same name as the column of the checked table. For example, implementations of the abstract elements “microcontroller for electronic components” are stored in *microcontrollers_for_electronic_components*. As a rule, there are requirements to the number of its pins and the amount of its flash memory:

```
"pins": 16,
"flash_kb": 540
```

In turn, the table *microcontrollers_for_electronic_components* has corresponding columns with the same names:

```
flash_kb INTEGER NOT NULL,
pins INTEGER NOT NULL,
```

It means that such requirements can be checked with an SQL query, where the name of the requirement is used as a column name, while the value of the requirements is used as a minimal value that satisfies the requirement.

Compatibility is also checked in accordance with the requirements that were formed during the design of the abstract system model. But those requirements were formed in accordance with abstract sub-elements. The information about which implementation from the corresponding database table is compatible with provided abstract sub-elements are stored in database tables, whose naming is following the rule: *db_table_name* + *“_and_compatibility”*.

For example, it is required to select the implementation of the “microcontroller for wireless communication” abstract element. Its implementations are stored in *mcs_for_ws_communication* (based on *selectable_elements_and_db_tables*). It means that information about implementations compatibility with abstract sub-elements must be stored in *mcs_for_ws_communication_and_compatibility*. Moreover, the structure of such tables must be as follows:

```
CREATE TABLE mcs_for_ws_communication_and_compatibility(
 id SERIAL PRIMARY KEY,
 option_id INTEGER REFERENCES mcs_for_ws_communication(id),
 requirement_id INTEGER REFERENCES abstract_subelements(id),
 UNIQUE (option_id, requirement_id)
);
```

Otherwise, the algorithm will not be able to check the compatibility requirements. And to continue the example about the “microcontroller for wireless communication”, in most cases, it must be compatible with the requirement to have the “bootloader” abstract sub-element, so its firmware can be updated.

Dependencies between abstract elements and links must be stored in database tables that are named based on the following rule:

option_1_table_name + “_and_” + option_2_table_name

For example, let’s consider dependencies of the “wireless signal receiver” abstract element, implementations of which are stored in *wireless_signal_receivers*. Selection of this element depends on the selection of implementations of the following abstract elements and links: the wireless link between charging stations and mobile robots for their communication during parking and “troyka shield”, to which this receiver is required to be connected. Possible implementations of the mentioned link are stored in *links_between_devices*, while ones for “troyka shield” — *troyka_shields*.

It means that dependencies between “wireless signal receiver” and abstract links must be stored in *wireless_signal_receivers_and_links_between_devices*, while with “troyka shield” in *wireless_signal_receivers_and_troyka_shields*. The structure of such tables of the database must be as follows:

```
CREATE TABLE wireless_signal_receivers_and_troyka_shields(
 id SERIAL PRIMARY KEY,
 option_1 INTEGER REFERENCES wireless_signal_receivers(id),
 option_2 INTEGER REFERENCES troyka_shields(id),
 UNIQUE (option_1, option_2)
);
```

Otherwise, the algorithm will not be able to take into account dependencies between implementations during their selection.

Once again, to not describe each insert to the developed database, its dump that was used for the design of the microcontroller-based physical security system, namely, perimeter monitoring system based on mobile robots, is available for download using the following link: https://github.com/levshun/PhD-mcbpss_design.

At this point, all tables of the database that are required for the correct work of the methodology for the design of microcontroller-based physical security systems, presented in Section 4.5, are manually filled with data that is required for the system, presented in Section 6.1.1. It means that the software implementation, presented in Chapter 5, can be executed as many times as necessary to check its compliance with the requirements, analyzed in Section 6.1.

It is important to note that while the fulfillment of the database with data for the design of one microcontroller-based physical security system requires a lot of time and effort, this effort can be used to design not only one system. It is assumed that the database would be filled in such a way that different microcontroller-based systems will partially share tasks, abilities, requirements, abstract elements, links and sub-elements as well as their implementations with each other, so the fulfillment of the database will take less time and effort after each system.

Moreover, while in this work many tables of the database that are responsible for the compatibility of elements of designed devices were filled manually, this process can be automated. Such automation means the process of checking parameters of elements that were checked for decision making (for example, voltage, current, interface, number of wires, etc.) during manual fulfillment.

In addition, based on the content of different online shops that are selling controllers and components for the implementation of microcontroller-based devices, it is possible to fill the database with information about such implementations automatically with the help of the parsing script. It would require multiple parsers, most likely one for each online shop, while each parser will require to be updated from time to time, because online shops are not static.

Finally, the process of fulfillment of the database presented in Section 5.2 can be shared between multiple enthusiasts and researchers. The formation of such a community would require the development of the web interface, which will guide contributors through the main steps of filling the database with data to design the selected microcontroller-based physical security systems. It would also require multiple algorithms for the checking of correctness of the provided data.

Thus, provided instructions on the fulfillment of the developed database should be considered as a general approach, which can be improved with help of the user-friendly interface as well as automated at many points, while the task of fulfillment can be shared among the community of enthusiasts.

6.3. Evaluation of the design methodology

To evaluate the methodology for the design of microcontroller-based physical security systems, presented in Chapter 4, it is required to analyze the compliance of its software implementation, presented in Chapter 5, with functional and non-functional requirements as well as to compare the obtained results with scientific and commercial solutions. In addition, it is planned to investigate the dependencies between the design time of the system, presented in Section 6.1.1, and parameters of the attacker, presented in Section 3.2.

6.3.1. Compliance with functional requirements

Functional requirements are representing the list of functions that are defining the actions that the software implementation must perform. In Section 1.4 the following functional requirements were formulated:

- building an abstract representation of designed systems;
- finding a trade-off between the resources spent and security;
- no restrictions on platforms and architectures of devices to be designed;
- extensibility of the design process;
- taking into account the physical layer of designed systems.

Let's consider compliance with each requirement in more detail.

Building an abstract representation of designed systems. The abstract representation of the designed system is provided by the abstract model. This model is constructed by the algorithm, presented in Section 4.3. This algorithm represents the system as an abstract hierarchical model that takes into account connections between system devices, their elemental composition, dependencies between device elements and requirements for them.

Finding a trade-off between the resources spent and security. The security of the designed system is based on the integration of security elements into its devices components composition. The number of security elements that are required to be integrated depends on the number of classes of attack actions that are possible on the designed device in accordance with its components composition, communication levels and parameters of the attacker against which the system is required to be protected, see Section 3.3. It means that the exact same microcontroller-based physical security system can be designed with different amounts of security elements if parameters of the attacker would differ.

No restrictions on platforms and architectures of devices to be designed. The software implementation firstly works with abstract elements, sub-elements and links and only after that replaces them with their implementations, see Section 4.5. It means that .

The extensibility of the design process. The structure of the database, presented in Section 5.2, contains tables, the content of which affects how microcontroller-based physical systems are designed by the software implementation. So, the first way to extend the developed design approach is to fill those tables with more data: additional examples of attack actions, security elements, tasks, abilities, requirements, elements, sub-elements, etc. Moreover, it is possible to use other models of the attacker and attack actions, if necessary: different number of parameters of the attacker as well as permissible ranges of their values, other classes of attack actions with different examples, etc. In addition, more parameters of elements can be taken into account as well as the list of calculated parameters for designed devices can be extended. Finally, additional algorithms can be integrated into the developed solution: design of software, formal verification, solution of optimization problems, design on the level of electronic components, etc.

Taking into account the physical layer of the designed systems. The software implementation designs microcontroller-based physical security systems in accordance with the extendable set-based hierarchical relational model, presented in Chapter 3. This model represents such systems as building blocks that are communicating with each other, while each block can have hardware and software elements. Moreover, communications between hardware and software elements are also taken into account as well as attack actions on them.

Thus, the software implementation meets all functional requirements.

6.3.2. Compliance with non-functional requirements

Non-functional requirements describe requirements and constraints imposed on the resources consumed by the software implementation. In Section 1.4, the set of non-functional requirements was divided into time consumption, resource consumption and validity. Let's consider them in more detail.

6.3.2.1. Time consumption

The method for the evaluation of time consumption is presented in Section 2.1, while this requirement is represented as follows:

$$P_T(TIME_N \leq TIME^{ACC}) \geq P_T^{ACC},$$

where $TIME_N$ — time required to design a secure system N; P_T — probability of designing a secure system in a given time frame; $TIME^{ACC}$— acceptable time for designing a secure system (1 sec for the abstract system model and 4 secs for the detailed model during the design process of the system of mobile robots for perimeter monitoring); P_T^{ACC}— acceptable probability value (0.99).

According to this method, time consumption is divided into time for designing the abstract and detailed models of the system, presented in Section 6.1.1.

Time consumption of the design process for the abstract system model is represented as the sum of the time consumption of each stage of this process:

$$TIME^{AM} = T_1^{AM} + T_2^{AM} + T_3^{AM}$$

where T_1^{AM} — time of the formation of requirements for the system and its devices; T_2^{AM} — time of formation of the component composition of the abstract model; T_3^{AM} — time of formation of the components hierarchy, connections between them, requirements for them, dependencies between them and recommendations for ensuring system security after implementation.

Time consumption of the design process for a detailed model of the system is the sum of the time consumption of each stage of this process:

$$TIME^{DM} = T_1^{DM} + T_2^{DM} + T_3^{DM}$$

where T_1^{DM} — time of the formation of selection steps for links between system devices and elements of the devices; T_2^{DM} — time of the selection of detailed elements of the system; T_3^{DM} — time of the calculation of device parameters as well as insertion of information about selected elements into the abstract model.

To obtain the average time consumption for each stage, the software implementation was executed 1000 times on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM. Time consumption was measured with the help of the time Python library. Based on such an experiment, the main time indicators of design stages for abstract and detailed models were obtained, see Tables 15 and 16, respectively.

Table 15. Time indicators for the design of the abstract system model

Stage	T_i^{min}, ms	T_i^{max}, ms	$T_i = \frac{3T_i^{min} + 2T_i^{max}}{5}$	$\sigma^2(T_i) = 0.4(T_i^{max} - T_i^{min})^2$
1	3.99	6.98	5.19	3.58
2	11.95	14.72	13.06	3.07
3	51.86	59.48	54.91	23.23
Total for the stage, ms			73.16	29.88

Then the value of the Laplace function $\Phi(Z)$, see Section 2.1, for $TIME^{ACC} = 1000\ ms$ for the design process of the abstract model of microcontroller-based physical security systems:

$$\Phi\left(\frac{TIME^{ACC} - \sum_{i=1}^{n} T_i}{\sqrt{\sum_{i=1}^{n} \sigma_i^2(T_i)}}\right) = \left(\frac{1000 - 73.16}{\sqrt{29.88}}\right) \simeq 169.56$$

Table 16. Time indicators for the design of the detailed system model

Stage	T_i^{min}, ms	T_i^{max}, ms	$T_i = \frac{3T_i^{min} + 2T_i^{max}}{5}$	$\sigma^2(T_i) = 0.4(T_i^{max} - T_i^{min})^2$
1	1.99	3.05	2.41	0.45
2	209.03	214.50	211.22	11.97
3	5.92	10.97	7.94	10.20
Total for the stage, ms			221.57	22.62

And for the design process of the detailed model of microcontroller-based physical security systems, the Laplace function $\Phi(Z)$ for $TIME^{ACC} = 4000\ ms$:

$$\Phi\left(\frac{TIME^{ACC} - \sum_{i=1}^{n} T_i}{\sqrt{\sum_{i=1}^{n} \sigma_i^2(T_i)}}\right) = \left(\frac{4000 - 221.57}{\sqrt{22.62}}\right) \simeq 794.45$$

Thus, according to the calculated values of the Laplace function, the probability of designing the abstract system model in a given time:

$$P_{NE}(TIME^{AM} \leq TIME_1^{ACC}) = 0.9999$$

And the probability of designing the detailed model of the system in a given time:

$$P_{NE}(TIME^{DM} \leq TIME_2^{ACC}) = 0.9999,$$

which means that the non-functional requirement for time consumption is satisfied.

6.3.2.2. Resource consumption

The method for the evaluation of resource consumption is presented in Section 2.2, while this requirement is represented as follows:

$$P_R(RES_N \leq RES^{ACC}) \geq P_R^{ACC}$$

where P_R — probability that the resources RES_N spent on the design process of microcontroller-based physical security system do not exceed the allowable value RES^{ACC}(0.25); P_R^{ACC}— acceptable value of probability (0.99).

According to this method, resource consumption is divided into particular indicators, namely, CPU, HDD and RAM. To obtain the average resource consumption for each indicator, the software implementation was executed 1000 times on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores, 16 threads) processor, 2 TB HDD and 32 GB RAM.

Resource consumption of each indicator was measured with the help of psutil Python library. Let's consider the results of such an experiment in more detail.

Resource consumption when using central processing unit (CPU):

$$RES_{CPU} = \frac{Q_{CPU}^{DM}}{Q_{CPU}^{ALL}}$$

where Q_{CPU}^{DM} — central processing unit time spent on the design process of microcontroller-based physical security systems; Q_{CPU}^{ALL} — total available CPU time.

During the design process of the microcontroller-based physical security system, each CPU core was loaded as follows:

- Core 1: thread 1 — 0.6%, thread 2 — 0.7%.
- Core 2: thread 3 — 6.7%, thread 4 — 0.0%.
- Core 3: thread 5 — 2.4%, thread 6 — 0.5%.
- Core 4: thread 7 — 1.1%, thread 8 — 0.0%.
- Core 5: thread 9 — 0.0%, thread 10 — 0.0%.
- Core 6: thread 11 — 0.3%, thread 12 — 0.2%.
- Core 7: thread 13 — 0.0%, thread 14 — 0.5%.
- Core 8: thread 15 — 1.2%, thread 16 — 0.0%.

Thus, the indicator $R_{CPU} = 0.0089$, which means that:

$$P_{RES}(RES_{CPU} \leq R^{ACC}) = 1,$$

and therefore requirement $P_{RES}(RES_{CPU} \leq R^{ACC}) \geq P_{RES}^{ACC}$ is fulfilled.

Resource consumption when using hard disk drive (HDD):

$$RES_{HDD} = \frac{Q_{HDD}^{DM}}{Q_{HDD}^{ALL}}$$

where Q_{HDD}^{DM} — HDD space used during the design process of microcontroller-based physical security systems; Q_{HDD}^{ALL} — total available HDD space.

The size of the software implementation of the design methodology for microcontroller-based physical security systems consists of sizes of:

- application: 27.1 MB;
- database: 15.0 MB.

Thus, the software implementation size is 42.1 MB and the hard drive utilization rate is $R_{HDD} = 0.000021$, which means that:

$$P_{RES}(RES_{HDD} \leq R^{ACC}) = 1,$$

and therefore requirement $P_{RES}(RES_{HDD} \leq R^{ACC}) \geq P_{RES}^{ACC}$ is fulfilled.

Resource consumption when using random-access memory (RAM):

$$RES_{RAM} = \frac{Q_{RAM}^{DM}}{Q_{RAM}^{ALL}}$$

where Q_{RAM}^{DM} — RAM amount used during the design of microcontroller-based physical security systems; Q_{RAM}^{ALL} — total available amount of RAM.

During the design process the additional 100290560 bytes of RAM was used, which means that $R_{RAM} = 0.0029$ and:

$$P_{RES}(RES_{RAM} \leq R^{ACC}) = 1,$$

and therefore requirement $P_{RES}(RES_{RAM} \leq R^{ACC}) \geq P_{RES}^{ACC}$ is fulfilled.

The obtained values of the corresponding indicators showed that each requirement $P_{RES}(r \leq R^{ACC}) \geq P_{RES}^{ACC}$ is fulfilled. It means that the non-functional requirement for resource consumption is satisfied.

6.3.2.3. Validity

The method for the evaluation of validity is presented in Section 2.3, while the number of parameters analyzed during the design process is selected as an indicator of the validity, namely number of: levels of the system, the security of which can be ensured; classes of attacks against which the system can be protected.

Levels of the system are divided into:

- $cn \leftrightarrow cr$ — controllers, components and their communications;
- $cr \leftrightarrow cr$ — controllers and their communications inside devices;
- $dv \leftrightarrow dv$ — devices and their communications with each other;
- $st \leftrightarrow st$ — system and its communications with other systems.

Classes of attack actions are divided into:

- cn — components and their communications with controllers;
- cr — controllers and their communications with other controllers;
- dv — devices and their communications with other devices;
- st — system and its communications with other systems.

Requirements for these indicators are set by comparison with existing systems and represented as follows:

$$\left|LEVELS_N\right| \geq max\left(\left|LEVELS_{s \in S}\right|\right),$$
$$\left|ATTACKS_N\right| \geq max\left(\left|ATTACKS_{s \in S}\right|\right),$$
$$\left|LEVELS_N \times ATTACKS_N\right| > max\left(\left|LEVELS_{s \in S} \times ATTACKS_{s \in S}\right|\right),$$

where N — suggested in this work design approach; S — set of design approaches with which N is compared; $\left|LEVELS_I\right|$ — number of levels of the system, the security of which can be ensured by the design approach I; $\left|ATTACKS_I\right|$ — number of classes of attack actions against which the system can be protected by the design approach I; $\left|LEVELS_I \times ATTACKS_I\right|$ — number of parameters that are analyzed during the design process by the approach I.

The software implementation of the methodology for the design of microcontroller-based physical security systems was compared with commercial and scientific solutions.

It is important to note that the comparison was made based on the publicly available data, where the presence or absence of the consideration of security of different levels of the system during the design process was determined.

The comparison with commercial solutions is based on levels of systems, the security of which can be ensured, see Table 17.

Table 17. The comparison with commercial solutions

		[117]	[119]	[120]	[122]	[124]	[125]	developed
level of the system	$cn \leftrightarrow cr$	–	–	–	–	–	–	+
	$cr \leftrightarrow cr$	–	–	–	–	–	–	+
	$dv \leftrightarrow dv$	–	–	–	–	–	–	+
	$st \leftrightarrow st$	+	+	+	+	+	+	+

The comparison showed that the developed design methodology takes into account security of all four levels of microcontroller-based systems, while commercial solutions are focused only on one of them — they are aiming to secure individual devices that can be connected to the cloud. It means that the indicator $\left|LEVELS_N\right| \geq max\left(\left|LEVELS_{s \in S}\right|\right)$ of the validity requirement is satisfied.

Another drawback of commercial solutions is that they are bound to the specific hardware, software, platforms and architectures. It means that if the designed system already contains devices whose hardware cannot be changed or there are restrictions that do not allow the use of suitable devices, then these solutions are not applicable. In addition, these solutions do not take into account the optimization of the system design process due to such limitations as parameters of the attacker, computational complexity, energy efficiency and price. It means that resulting systems may not be reasonable for a developed use case because of no trade-off between resources and the provided security level.

The comparison with scientific solutions is based on classes of attack actions against which the system can be protected, see Table 18.

Table 18. The comparison with scientific solutions

		[72]	[84]	[115]	[116]	[128]	[129]	developed
classes of attack actions	cn	–	–	+	–	+	–	+
	cr	*	*	+	+	+	+	+
	dv	*	*	–	–	–	+	+
	st	–	–	–	–	–	–	+

Once again, it is important to note that the comparison was made based on the publicly available data, where the presence or absence of the consideration of protection against different classes of attack actions was determined. In addition,

note that "*" for [72] and [84] means that the models and approaches used by the authors can be improved for taking the corresponding classes of attack actions into account during the design process, while such functionality was not presented.

The comparison showed that the developed design methodology provides protection against all four classes of attack actions on microcontroller-based systems, while scientific solutions are considering two of them at most. It means that the indicator $\left|ATTACKS_N\right| \geq max\left(\left|ATTACKS_{s \in S}\right|\right)$ of the validity requirement is also satisfied.

The drawback of existing scientific solutions is that they are focused on certain aspects of security, which ensures their inapplicability for providing the security of designed systems in general. For example, some approaches do not take into account that the functionality of system components is determined not only by software, but also by hardware. Other approaches are considering designed devices in isolation from the system they will work in. It means that not all security aspects are taken into account and the security of the system as a whole will not be ensured. Also, some techniques are aiming at ensuring the security of communications between devices. The drawback is that such techniques provide secure connection between designed systems and external systems only from the designed side, which can lead to security issues during the design of complex multi-level systems.

The last indicator of the validity requirement to be checked is the number of parameters that are taken into account during the design process of microcontroller-based physical security systems. And according to the object function, see Section 1.5, it is required to not only surpass other solutions in the number of analyzed parameters, but also to maximize them.

Results of analysis of commercial and scientific solutions in accordance with the number of levels of the system, the security of which can be ensured, and the number of classes of attacks against which the system can be protected are presented in Table 19.

Table 19. The comparison with all solutions

Solutions	Levels of the system				Classes of attack actions			
	$cn{\leftrightarrow}cr$	$cr{\leftrightarrow}cr$	$dv{\leftrightarrow}dv$	$st{\leftrightarrow}st$	cn	cr	dv	st
[72]	–	–	–	–	–	*	*	–
[84]	–	–	–	–	–	*	*	–
[115]	+	+	–	–	+	+	–	–
[116]	+	+	–	–	–	+	–	–
[117]	–	–	–	+	–	–	+	+

Solutions	Levels of the system				Classes of attack actions			
	$cn{\leftrightarrow}cr$	$cr{\leftrightarrow}cr$	$dv{\leftrightarrow}dv$	$st{\leftrightarrow}st$	cn	cr	dv	st
[119]	–	–	–	+	–	–	+	+
[120]	–	–	–	+	–	–	+	+
[122]	–	–	–	+	–	–	–	+
[124]	–	–	–	+	–	–	+	+
[125]	–	–	–	+	–	–	–	+
[128]	+	+	–	–	+	+	–	–
[129]	–	–	+	–	–	+	+	–
developed	+	+	+	+	+	+	+	+

The comparison showed that the developed design methodology provides protection against all analyzed classes of attack actions as well as takes into account security of all analyzed levels of microcontroller-based physical security systems, while other solutions are not considering that many parameters. Once again, note that "*" for [72] and [84] means that provided models and approaches can be improved for taking the corresponding classes of attack actions into account. It means that the last indicator $\left|LEVELS_N \times ATTACKS_N\right| > max\left(\left|LEVELS_{s \in S} \times ATTACKS_{s \in S}\right|\right)$ is satisfied, thus, the non-functional requirement for validity is satisfied as well.

6.3.3. Dependencies between design time and parameters of attackers

The model of the attacker, presented in Section 3.2, characterizes his or her capabilities in accordance with three parameters: ac — type of access, kn — type of knowledge, rs — type of resources, where:

- ac can be in range between 1 and 5 and describes the type of access the attacker has to the system (for example, physical access to system devices);
- kn — 1 and 4 and describes the amount of information available about the system (for example, system hardware and software are known);
- rs — 1 and 3 and describes the number of resources available to the attacker (for example, the attacker can use specialized software tools).

To obtain the average time for each combination of values of parameters, the software implementation was executed 10 times for each combination on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM. Time consumption was measured with the help of the time Python library.

The relationship between these parameters and dt — system design time is shown in Figure 113. The scale on the left from 0 to 5 reflects changes in values of the attacker's parameters — ac, kn and rs are shown as area charts, while the scale on the right from 0 to 0.12 reflects the design time — dt is shown as a black line.

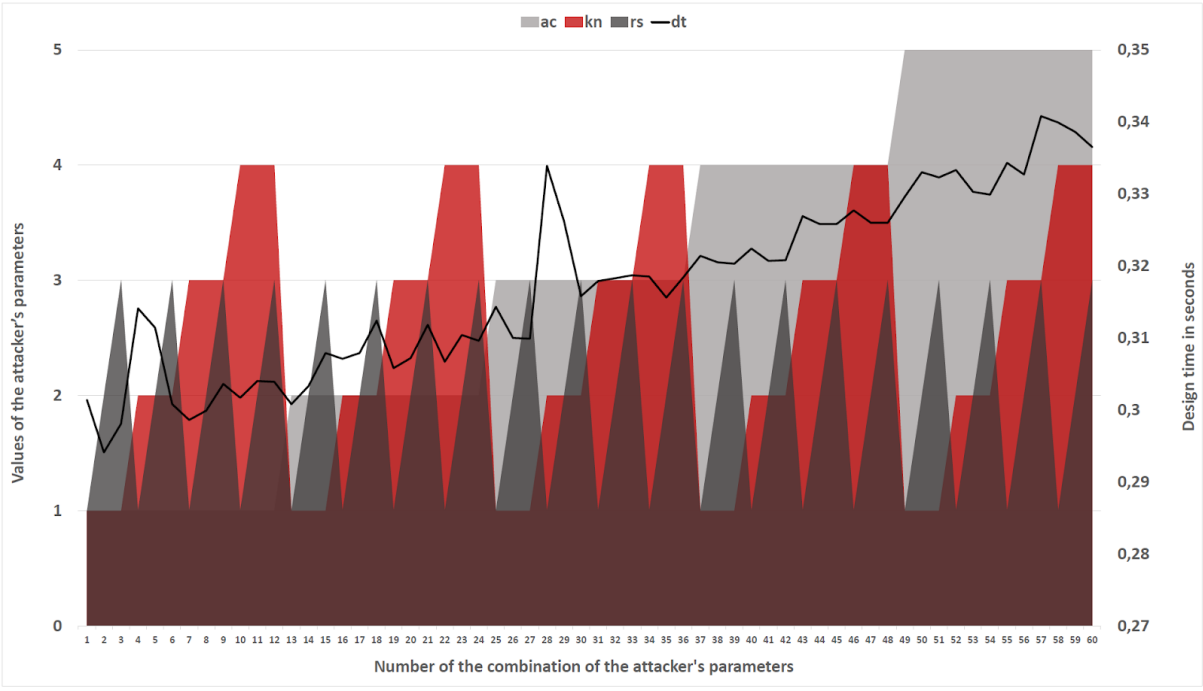

Figure 113. Dependencies between design time and parameters of attackers

The minimum design time was 0.2941 seconds, while the maximum — 0.3408.

According to the related work analysis, presented in Chapter 1, there is no data available about the average time of design of microcontroller-based physical security systems by commercial or scientific solutions with which the developed one was compared in the previous section. Moreover, even if this data would be available, it is difficult to compare design approaches when different systems with different amounts of devices that contain different amounts of elements are designed.

6.4. Discussion

Experimental evaluation of the software implementation of the methodology for the design of microcontroller-based physical security systems showed that the developed approach has advantages in comparison with analogs. In this work a new approach to the design is presented, which allows combining various design techniques on the basis of hierarchical relational model transformation algorithms. This approach is modular and extensible, takes into account the security of the physical layer of microcontroller-based systems, works with the abstract system representation and is looking for a trade-off between the security of the final solution

and resources expended on it. Moreover, the methodology has a strong focus on security and aims at ensuring the protection of designed systems against various attack actions at early stages of their lifecycle, considers security components as an integral part of the system and checks if the system can be designed in accordance with given requirements and limitations.

The results obtained in this work are very important for solving fundamental issues in the field of ensuring information security of microcontroller-based systems and are aimed at expanding and improving the existing model-methodological apparatus associated with the design of such systems. The practical significance of the results lies in the fact that the system based on the proposed models, algorithms and methodology can be used as a tool for designing secure systems, thus, avoiding errors in the early stages of their life cycle.

The following can be identified as potential consumers of the results of this work: scientific community, commercial companies, educational organizations as well as users and administrators of systems microcontroller-based systems, especially physical security ones. Let's consider each potential customer in more detail.

The scientific community may be interested in expanding and improving the developed model-methodological apparatus as well as forming a community of enthusiasts to fill and maintain the database of the obtained solution.

Commercial companies may be interested in introducing the developed model-methodological apparatus into the lifecycle of microcontroller-based systems they develop to improve their security.

Educational organizations may be interested in conducting lectures and practical classes to share the experience gained in this work in the design of microcontroller-based physical security systems.

Users and administrators can use the results obtained in this work to gain an understanding of the security status of microcontroller-based devices and systems that are within their area of responsibility.

It is important to note that the results of this work can be brought to practical use in the form of a software product. The use of such a product will help to reduce the number of weak places and architectural defects in microcontroller-based systems, thereby significantly reducing their attack surface. In turn, this will reduce the security risks that can lead to financial losses, loss of time as well as the safety of people, which ensures the relevance and high significance of this work.

This work presents not only the developed model-methodological apparatus for the design of microcontroller-based physical security systems with its software implementation, but also a framework that can be improved in various ways.

For example, it can be improved with the use of genetic algorithms during the automated selection of implementations of different components and controllers among options that are satisfying given requirements. Based on priorities of parameters like price, energy consumption and computation efficiency it would be possible to solve the optimization task to find reasonable component compositions.

In addition, the verification process can become an integral part of the solution. It can provide the formal check of the possibility to design microcontroller-based physical security systems in accordance with the given requirements. Moreover, it can provide the formal check of the security level of the designed system in accordance with the model of the attacker.

Finally, the use of a component-based approach to modeling microcontroller-based physical security systems can be extended with semi-natural, simulation and analytical modeling. The advantage of integration of these approaches is in the possibility to represent various aspects of such systems, including dynamic ones.

6.5. Conclusions on Chapter 6

Experimental evaluation of the methodology for the design of microcontroller-based physical security systems is done with help of its software implementation, which consists of the Python script, PostgreSQL database and Tkinter interface.

The developed application was executed on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM.

For the experiment, it was decided to design a microcontroller-based physical security system that provides perimeter monitoring based on mobile robots. Such a system was chosen due to the presence of several types of devices, multiple communications between them, as well as the need to use many different elements for each device in the system (server consists of 8 elements with sub-elements, station — 12 and robot — 17, which means that such a system is appropriate in accordance with the provided requirements).

The process of modeling this system in terms of the extendable set-based hierarchical relational model, see Chapter 3, is presented in Appendix A.

Based on links between tasks, abilities and requirements of this system, it was concluded that such a system consists of the following types of devices: the server,

charging stations and mobile robots. In turn, devices are having their own tasks that are linked with their abilities and requirements, while data about all of them must be stored in corresponding tables of the database.

The process of manual fulfillment of all tables of the developed database that are required for the correct work of the software implementation was described. And after this process is done, the software implementation can be executed as many times as necessary to check its compliance with the requirements.

It is important to note that while the fulfillment of the database with data about one system requires a lot of time and effort, this effort can be used to design other systems as well. The database can be filled in such a way that different systems will partially share tasks, abilities, requirements, abstract elements, links and sub-elements as well as their implementations with each other, so the fulfillment of the database will take less time and effort for every next system.

Moreover, while in this work many tables of the database that are responsible for the compatibility of elements of designed devices were filled manually, this process can be automated. For example, based on the content of different online shops that are selling controllers and components for the implementation of microcontroller-based devices, it is possible to fill the database with information about such implementations automatically with the help of the parsing script.

The software implementation satisfied all functional requirements and was analyzed in terms of compliance with non-functional ones, see Table 20.

Table 20. Results of analysis of the software implementation

	Result	Conclusion
time consumption	$P_{NE}(TIME^{AM} \le TIME_1^{ACC}) = 0.9999,$ $P_{NE}(TIME^{DM} \le TIME_2^{ACC}) = 0.9999$	✓
resource consumption	$P_{RES}(RES_{CPU} \le R^{ACC}) = 1,$ $P_{RES}(RES_{HDD} \le R^{ACC}) = 1,$ $P_{RES}(RES_{RAM} \le R^{ACC}) = 1$	✓
validity	$\lvert LEVELS_N \rvert \ge max(\lvert LEVELS_{s \in S} \rvert),$ $\lvert ATTACKS_N \rvert \ge max(\lvert ATTACKS_{s \in S} \rvert),$ $\lvert LEVELS_N \times ATTACKS_N \rvert > max(\lvert LEVELS_{s \in S} \times ATTACKS_{s \in S} \rvert)$	✓
objective function	$O_F: \lvert LEVELS_N \times ATTACKS_N \rvert \to max$	✓

It means that the goal of this work is reached, namely, the development of the design methodology for microcontroller-based physical security systems that builds an abstract representation of designed systems, finds a trade-off between the resources spent and ensured level of security, has no restrictions on platforms and architectures of the devices to be designed, is extensible and takes into account the physical layer of designed systems, while being in compliance with the requirements for time and resource consumption and taking into account more security parameters of designed systems than analogs.

In addition, the dependencies between the design time of the system and parameters of the attacker (types of access — from 1 to 5, knowledge — from 1 to 4 and resources — from 1 to 3) were investigated. Those parameters are defining attack actions that are possible for the attacker, which means that the amount of security elements integrated into the designed system differs from one combination of parameters to another. During the experiments, the minimum design time was 0.2941 seconds, while the maximum — 0.3408.

It is important to note that the developed solution is not aimed to replace experts in security of microcontroller-based systems. It is understandable that in most situations, experts are aware of existing best practices and highly specialized solutions and are able to design such systems at a very high level, while the quality of the solution provided by the implementation of the methodology directly depends on the correctness and completeness of its database. But even for an expert, it can be useful in terms of automatization of routine tasks as well as at offering options that are different from those familiar to him or her.

Moreover, this work presents not only the developed model-methodological apparatus for the design of microcontroller-based physical security systems, but also a framework that can be improved in various ways: use of genetic algorithms during the automated selection of implementations, formal check of the possibility to design microcontroller-based, etc.

For example, we already achieved some results in the process of verification of microcontroller-based devices, see Appendix B.

In addition, we achieved some results in the connection of implementations of components, controllers, their software and firmware with vulnerabilities, described in CVE (Common Vulnerabilities and Exposures) format, see Appendix C.

The dump of the database that was used for the design of the microcontroller-based physical perimeter monitoring system based on mobile robots, is available for download using the following link: https://github.com/levshun/PhD-mcbpss_design.

Conclusion

The goal of this work is to develop the design methodology for microcontroller-based physical security systems that builds an abstract representation of designed systems, finds a trade-off between the resources spent and ensured level of security, has no restrictions on platforms and architectures of the devices to be designed, is extensible and takes into account the physical layer of designed systems, while being in compliance with the requirements for time and resource consumption and taking into account more security parameters of designed systems than analogs.

The process of achieving this goal is described in 6 chapters:

1. Systematic analysis of the main issues of ensuring the information security of microcontroller-based systems.
2. Methods for the evaluation of the design methodology for microcontroller-based physical security systems.
3. The extendable set-based hierarchical relational model of microcontroller-based physical security systems.
4. Algorithms and methodology for the design of microcontroller-based physical security systems.
5. Software implementation of the methodology for the design of microcontroller-based physical security systems.
6. Experimental evaluation of the methodology for the design of microcontroller-based physical security systems.

Let’s consider the main results of each chapter in more detail.

In Chapter 1 the analysis and systematization of modern research in the field of information security of microcontroller-based systems have been carried out. Place and role of design techniques in ensuring the information security of such systems was shown. The drawbacks of existing solutions were pointed out. It was concluded that a general approach for solving the issue of designing secure systems is not done yet. Among all possible systems, in this work only physical security systems were chosen as an area of the application, because in such systems it is required to ensure not only the functionality of the system, but also to ensure its security against cyber-physical attacks during the design process. The developed architecture of microcontroller-based physical security systems contains components, controllers and devices that are communicating with each other, their software and firmware. The functional and non-functional requirements for the new design methodology were formulated. The objective function of the developed design methodology is aimed at maximization of the number of security parameters that are analyzed during the design process of microcontroller-based systems.

Chapter 2 presents methods for the evaluation of the design methodology for microcontroller-based physical security systems. Methods are divided into evaluation of time and resource consumption as well as validity. Time consumption is defined as the probability that the approach is able to design a system in accordance with the input data in a given time frame. Resource consumption — probability that the number of resources (CPU, HDD, RAM) used during the design process will not exceed the allowable value. And validity — number of levels of the system, the security of which can be ensured, and number of classes of attack actions against which the system can be protected, are maximized.

In Chapter 3 one of the main findings of this work is presented, namely, the extendable set-based hierarchical relational model of microcontroller-based physical security systems. For this model, a component-based approach was chosen as the most detailed way of representation. Moreover, this approach is the most appropriate one if it is required to take into account the security of the system as early as possible. Developed in this work model represents microcontroller-based physical security systems as building blocks (hardware and software elements) that are communicating with each other through links (protocols and interfaces), while the security is taken into account in accordance with models of the attacker and attack actions. Within the framework of the developed model, all elements are connected with each other through their properties. The developed model of attacker is distinguishing attackers according to their types of access, knowledge and resources, while the developed model of attack actions distinguishes attacks based on their subject, object and impact method. In addition, instead of separate impact methods, it was decided to use classes of attack actions, while each class contains multiple examples of methods.

Chapter 4 presents the following two main findings of this work, namely, the set of algorithms and the methodology for the design of microcontroller-based physical security systems. Let’s consider each algorithm and methodology in more detail.

The algorithm for the formation of requirements for microcontroller-based physical security systems is used to extract attack actions that are possible for the attacker as well as the list of devices of the designed system, their links, communications, bases and requirements in accordance with general tasks of the system. The output data of the algorithm is well-structured and JSON-based. The work process of the algorithm is automatic, the operator is required for the transformation of wishes of the stakeholder into general tasks of the system and parameters of the attacker.

The algorithm for the formation of microcontroller-based physical security systems component compositions is used to extract abstract elements and sub-elements of devices of the system, security recommendations to the system and its devices implementations as well as abstract links between devices with related to them abilities based on attack actions that are possible for the attacker, list of devices of

the system, their bases, types of communications and links, requirements for them. This algorithm works with abstract elements, links and recommendations and represents the components compositions as multiple devices, each of which has multiple abstract elements, while each abstract element can have multiple abstract sub-elements. The work process of the algorithm is automated, the operator is not required. Its output data is also well-structured and JSON-based.

The algorithm for the design of abstract models of microcontroller-based physical security systems is used to construct an abstract representation of the designed system based on its devices list, their abilities, elements and sub-elements as well as security recommendations. This algorithm represents the system as an abstract hierarchical model that takes into account connections between system devices, their elemental composition, dependencies between device elements and requirements for them. As output data, the algorithm provides the abstract system model that contains abstract system representation. The structure of the abstract system model is JSON-based and contains the following fields: devices, recommendations and links, while each element of the device from the “components” field has its own components (sub-elements), links, requirements and dependencies. The work process of the algorithm is automatic, the operator is not required.

The algorithm for the design of detailed models of microcontroller-based physical security systems is used to construct a detailed representation of the designed system based on its abstract representation. Detailed model of the system preserves and expands the structure of the abstract model of the system and takes into account compatibility, requirements, dependencies and hierarchy of its elements. The process of transition from the abstract system model to the detailed one is a step-by-step process. Each step represents the selection of the concrete implementation of one of the system elements, while the sequence of steps is formed in accordance with hierarchy and dependencies between those elements. As output data, the algorithm provides the detailed system model. Its structure is also JSON-based. Moreover, it has the same structure as the abstract system model but with some additions: each element from the components field that was selected is extended with the selected field; each device of the system is extended with the parameters field; each link between devices of the system is extended with the selected field. The work process of the algorithm is mostly automated, involvement of the operator is possible at the stage of selection of the concrete implementations of elements among suitable options provided by the algorithm. Alternatively, the algorithm can select concrete implementations on its own.

The methodology for the design of microcontroller-based physical security systems consists of two main cycles. The main goal of the first cycle is to design the abstract system model based on provided requirements, while the second one is about the design of the detailed system model based on selection of components. Each cycle of the methodology consists of the testing process and seven stages that are

associated with the extendable set-based hierarchical relational. The testing process occurs after each stage as many times as necessary to build the model of the system. The objective of the testing process is in checking constructed models in terms of their correctness and compatibility. In terms of the input data, the first cycle works with requirements and limitations, while providing abstract models of system elements and the abstract model of the system as an output. In its turn, the second cycle works with models that were designed by the first cycle and adds to the abstract model data about selected devices and their parameters as an output. Another way to represent the workflow of the methodology, is to showcase its connection with presented algorithms: first three algorithms are representing the abstract system model design cycle, while the last one is representing the detailed system model design cycle.

In Chapter 5 one of the main findings of this work, namely, the software implementation of the methodology for the design of microcontroller-based physical security systems is presented. Its architecture consists of the Python script, PostgreSQL database and Tkinter interface. PostgreSQL database is required to store data about the presented extendable set-based hierarchical relational model, as well as data for algorithms and methodology. Python script represents the implementation of the algorithms and methodology: each algorithm is implemented as a number of functions, while all functions are connected with each other in a single methodology. Tkinter interface is required to receive input data from the operator, namely, parameters of the attacker and tasks of the designed system, as well as to provide the output data to him or her.

Chapter 6 presents the experimental evaluation of the methodology for the design of microcontroller-based physical security systems. It is done with the help of the presented software implementation of the methodology. For the experiment, it was decided to design a system that provides perimeter monitoring based on mobile robots. Such a system was chosen due to the presence of several types of devices, multiple communications between them, as well as the need to use many different elements for each device in the system (server consists of 8 elements with sub-elements, station — 12 and robot — 17, which means that it is satisfies the provided requirements). The process of modeling this system in terms of the extendable set-based hierarchical relational model is presented in Appendix A.

The process of manual fulfillment of all tables of the developed database that are required for the correct work of the software implementation was described. It is important to note that while the fulfillment of the database with data about one system requires a lot of time and effort, this effort can be used to design other systems as well. The database can be filled in such a way that different systems will partially share tasks, abilities, requirements, abstract elements, links and sub-elements as well as their implementations with each other, so the fulfillment of the database will take less time and effort for every next system.

Moreover, while in this work many tables of the database that are responsible for the compatibility of elements of designed devices were filled manually, this process can be automated. For example, based on the content of different online shops that are selling controllers and components for the implementation of microcontroller-based devices, it is possible to fill the database with information about such implementations automatically with the help of the parsing script. Such a functionality represents one of the future work directions of this work. For example, it is possible to use artificial intelligence methods for this task.

In addition, with the help of the user-friendly interface, the task of fulfillment of the database can be shared among the community of enthusiasts. Such a functionality represents one of the future work directions of this work. For example, it is possible to use the Django web framework to provide such an interface.

It might also be interesting to investigate other technologies for the storage of the extendable set-based hierarchical relational model of microcontroller-based physical security systems. For example, object-oriented models like UML (Unified Modeling Language) with its extensions for Internet of Things or ontologies like OWL (W3C Web Ontology Language).

The software implementation was executed on the computer with Windows 10 x64 operating system, Intel Core i7-4790 CPU 3.60GHz (8 cores) processor, 2 TB HDD and 32 GB RAM. Experiments showed that it satisfies all functional and non-functional requirements, provided during the research problem statement.

It means that the goal of this work is reached, namely, the development of the design methodology for microcontroller-based physical security systems that builds an abstract representation of designed systems, finds a trade-off between the resources spent and ensured level of security, has no restrictions on platforms and architectures of the devices to be designed, is extensible and takes into account the physical layer of designed systems, while being in compliance with the requirements for time and resource consumption and taking into account more security parameters of designed system than commercial and scientific analogs.

In addition, the dependencies between the design time of the system and parameters of the attacker were investigated. Those parameters are defining attack actions that are possible for the attacker, which means that the amount of security elements integrated into the designed system differs from one combination of parameters to another. During the experiments, the minimum design time was 0.2941 seconds, while the maximum — 0.3408.

It is important to note that the developed solution is not aimed to replace experts in security of microcontroller-based systems. It is understandable that in most situations, experts are aware of existing best practices and highly specialized

solutions and are able to design such systems at a very high level, while the quality of the solution provided by the implementation of the methodology directly depends on the correctness and completeness of its database. But even for an expert, it can be useful in terms of automatization of routine tasks as well as at offering options that are different from those familiar to him or her.

Moreover, this work presents not only the developed model-methodological apparatus for the design of microcontroller-based physical security systems, but also a framework that can be improved in various ways: use of genetic algorithms during the automated selection of implementations, formal check of the possibility to design microcontroller-based systems, etc. For example, we already achieved some results in the process of verification of microcontroller-based devices, see Appendix B.

In Section 4.5 it was mentioned that the developed methodology might find out that selected components are not working as intended — some of the requirements or limitations are violated. And in such a situation, the methodology will try to rebuild the system model until it is done or conclude that it is not possible — the number of attempts is limited. This functionality currently is not developed in the software implementation, see Chapter 5, and represents one of the future work directions.

It is also possible to connect implementations of components and controllers as well as their software and firmware with vulnerabilities, described in CVE (Common Vulnerabilities and Exposures) format. Such a connection is possible based on CPE (Common Platform Enumeration) descriptions, more precisely, on their URIs (Uniform Resource Identifier). So, if the information about the vendor, product and version of the hardware, application or operating system is available, with some probability of false positives, it is possible to extract the corresponding CPE URIs. The combination of CPE URIs of the device represents its configuration. Such configuration can be checked in terms of being vulnerable and connected to CVEs. The results achieved in this direction are presented in Appendix C.

References

1. Desnitsky V., Chechulin A., Kotenko I., Levshun D. and Kolomeec M. Combined design technique for secure embedded devices exemplified by a perimeter protection system // Trudy SPIIRAN. 2016. Vol. 48. P. 5-31. [in Russian]
2. Levshun D., Chechulin A., Kotenko I. and Chevalier Y. Design and verification methodology for secure and distributed cyber-physical systems // 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 2019. P. 1-5.
3. Pressley A. Securing connections in the cloud and across IoT devices // Intelligent CIOEurope, 2020.
4. Baheti R., Gill H. Cyber-physical systems // The impact of control technology. 2011. Vol. 12. No. 1. P. 161-166.
5. Schwab K. The fourth industrial revolution. Currency, 2017.
6. Hehenberger P., Vogel-Heuser B., Bradley D., Eynard B., Tomiyama T. and Achiche S. Design, modelling, simulation and integration of cyber physical systems: Methods and applications // Computers in Industry. 2016. Vol. 82. P. 273-289.
7. Zegzhda D. Sustainability as a criterion for information security in cyber-physical systems // Automatic control and computer sciences. 2016. Vol. 50. No. 8. P. 813-819.
8. Broy M. Engineering cyber-physical systems: challenges and foundations // Complex Systems Design & Management. Springer, Berlin, Heidelberg, 2013. P. 1-13.
9. Li Y., Li X., Wang L. and Li Y. Limestone-gypsum wet flue gas desulfurization based on Cyber-Physical System // 2019 Chinese Control And Decision Conference (CCDC). IEEE, 2019. P. 473-477.
10. Rogozinsky G. Multi-domain approach and models of cyber-physical objects in information representation systems // Proceedings of Telecommunication Universities. 2017. Vol. 3. No. 4. P. 88-93.
11. Xiao-Le W., Hong-Bin H., Su D. and Li-Na C. A service-oriented architecture framework for cyber-physical systems // Recent Advances in Computer Science and Information Engineering. Springer, Berlin, Heidelberg, 2012. P. 671-676.
12. Dong P., Han Y., Guo X. and Xie F. A systematic review of studies on cyber physical system security // International Journal of Security and Its Applications. 2015. Vol. 9. No. 1. P. 155-164.
13. Lee J., Bagheri B., Kao H. A cyber-physical systems architecture for industry 4.0-based manufacturing systems // Manufacturing letters. 2015. Vol. 3. P. 18-23.
14. Xia X., Liu C., Wang H. and Han Z. A Design of Cyber-Physical System Architecture for Smart City // Recent Trends in Intelligent Computing, Communication and Devices. Springer, Singapore, 2020. P. 967-973.

15. Rojas R., Rauch E., Vidoni R. and Matt D. Enabling connectivity of cyber-physical production systems: a conceptual framework // Procedia Manufacturing. 2017. Vol. 11. P. 822-829.

16. Alguliyev R., Imamverdiyev Y., Sukhostat L. Cyber-physical systems and their security issues // Computers in Industry. 2018. Vol. 100. P. 212-223.

17. Cardin O. Classification of cyber-physical production systems applications: Proposition of an analysis framework // Computers in Industry. 2019. Vol. 104. P. 11-21.

18. Zegzhda D., Poltavtseva M., Lavrova D. Systematization and security assessment of cyber-physical systems // Automatic control and computer sciences. 2017. Vol. 51. No. 8. P. 835-843.

19. Romanov V.N. Approach for complex systems analysis. SPb: SZTU. 2011. 287 p. [in Russian]

20. Kohanovskiy V., Sergeyeva M., Komakhidze M. System complexity index. Vestnik Donskogo gosudarstvennogo tekhnicheskogo universiteta – Advanced Engineering Research. 2012. Vol. 4(65). P. 22–26. [in Russian]

21. Burg A., Chattopadhyay A., Lam K. Wireless communication and security issues for cyber–physical systems and the Internet-of-Things // Proceedings of the IEEE. 2017. Vol. 106. No. 1. P. 38-60.

22. Mikhaylov K., Tervonen J. Evaluation of power efficiency for digital serial interfaces of microcontrollers // 2012 5th International Conference on New Technologies, Mobility and Security (NTMS). IEEE, 2012. P. 1-5.

23. Avatefipour O., Hafeez A., Tayyab M. and Malik H. Linking received packet to the transmitter through physical-fingerprinting of controller area network // 2017 IEEE Workshop on Information Forensics and Security (WIFS). IEEE, 2017. P. 1-6.

24. Gaifulina D., Kotenko I., Fedorchenko A. A Technique for Lexical Markup of Structured Binary Data for Problems of Protocols Analysis in Uncertainty Conditions. Sistemy upravleniya, svyazi i bezopasnosti — Systems of Control, Communication and Security. 2019. Vol. 4. P. 280–299. [in Russian]

25. Doynikova E. [Security assessment and selection of protective measures in computer networks based on attack graphs and service dependencies]. Dissertaciya na soiskanieuchenoj stepeni kandidata tekhnicheskih nauk – Dissertation for the degree of Candidate of Technical Sciences. 2017. 207 p. [in Russian]

26. Federal’nyj zakon “O bezopasnosti kriticheskoj informacionnoj infrastruktury Rossijskoj Federacii” ot 26.07.2017 No 187-FZ (poslednyaya redakciya) – Federal Law “On the Security of the Critical Information Infrastructure of the Russian Federation” dated July 26, 2017 No. 187-FZ (last edition). Consultant Plus. [in Russian]

27. Stallings W. The internet of things: network and security architecture // Internet Protoc. J. 2015. Vol. 18. No. 4. P. 2-24.

28. Khaitan S. and McCalley J. Design techniques and applications of cyberphysical systems: A survey // IEEE Systems Journal. 2014. Vol. 9. No. 2. P. 350-365.

29. Gomez C., Chessa S., Fleury A., Roussos G. and Preuveneers D. Internet of Things for enabling smart environments: A technology-centric perspective // Journal of Ambient Intelligence and Smart Environments. 2019. Vol. 11. No. 1. P. 23-43.

30. Monostori L. Cyber-physical production systems: Roots, expectations and R&D challenges // Procedia Cirp. 2014. Vol. 17. P. 9-13.

31. Gurjanov A., Zakoldaev D., Zharinov I., Nechaev V. Design concepts for digital project and production companies of Industry 4.0 standard. Nauchno-tekhnicheskij vestnik informacionnyh tekhnologij, mekhaniki i optiki — Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2018. Issue 18. Vol. 3. P. 421-427. [in Russian]

32. Nikolakis N., Maratos V., Makris S. A cyber physical system (CPS) approach for safe human-robot collaboration in a shared workplace // Robotics and Computer-Integrated Manufacturing. 2019. Vol. 56. P. 233-243.

33. Liu H., Wang L. Remote human–robot collaboration: A cyber–physical system application for hazard manufacturing environment // Journal of manufacturing systems. 2020. Vol. 54. P. 24-34.

34. Levin B., Rosenberg I., Tsvetkov V. Transport cyber-physical systems. Nauka i tekhnologii zheleznyh dorog — Science and technology of railways. 2017. Issue 3. Vol. 3. P. 3. [in Russian]

35. Volkov A. Cybernetics of construction systems. Promyshlennoe i grazhdanskoe stroitel'stvo — Cyber-physical construction systems. 2017. Vol. 9. P. 4–7. [in Russian]

36. Dey N., Ashour A., Shi F., Fong S. and Tavares J. Medical cyber-physical systems: A survey // Journal of medical systems. 2018. Vol. 42. No. 4. P. 1-13.

37. Shishvan O., Zois D., Soyata T. Incorporating Artificial Intelligence into Medical Cyber Physical Systems: A Survey // Connected Health in Smart Cities. Springer, Cham, 2020. P. 153-178.

38. Popov D.S. [Information support for technological preparation of repair production in transport]. Vestnik Sibirskogo gosudarstvennogo universiteta putej soobshcheniya — Journal of the Siberian State Transport University. 2007. Vol. 17. P. 163–168. [in Russian].

39. Fedorchenko A., Doynikova E., Kotenko I. Automated detection of assets and calculation of their criticality for the analysis of information system security. Trudy SPIIRAN — SPIIRAS Proceedings. 2019. Issue 18(5). P. 1182–1211. [in Russian]

40. Koptenkov M. Information categorization is the first step to ensuring the information security of an organization. Bezopasnost Informatsionnykh Tekhnologiy — IT Security. 2011. Issue 18. Vol. 4. P. 117–119. [in Russian]

41. Mikoni S. Model of the participants in the life cycle of a socio-cyber-physical system. Tekhnologicheskaya perspektiva v ramkah evrazijskogo prostranstva: novye rynki i tochki ekonomicheskogo rosta – Technological perspective within the Eurasian space: newmarkets and points of economic growth. 2019. pp. 341–347. [in Russian]

42. GOST R. 53114-2008 Information security. Ensuring information security in the organization. Basic terms and definitions. 2008. [in Russian]

43. The basic model of threats to the security of personal data during their processing in personal data information systems. Federal Service for Technical and Export Control(FSTEC of Russia), February 15, 2008. [in Russian]

44. Metodika opredeleniya ugroz bezopasnosti informacii v informacionnyh sistemah [Methodology for determining threats to information security in information systems]. Federal'naya sluzhba po tekhnicheskomu i eksportnomu kontrolyu (FSTEK Rossii), proekt, 2015. [in Russian]

45. Methodological recommendations for the development of regulatory legal acts that determine threats to the security of personal data, relevant when processing personal data in information systems of personal data used in the implementation of relevant activities.Federal Security Service (FSB of Russia), March 31, 2015, No. 149/7/2/6-432. [in Russian]

46. Rocchetto M., Tippenhauer N. On attacker models and profiles for cyber-physical systems // European Symposium on Research in Computer Security. Springer, Cham, 2016. P. 427-449.

47. Desnitsky V. A Modeling and Analysis of Security Incidents in a Cyber-Physical System for Water Supply Management. Informacionnye tekhnologii i telekommunikacii – Telecom IT. 2017. Vol. 5. No. 3. pp. 93–102. [in Russian]

48. GOST R. ISO/IEC 27000–2012 Information technology. Security methods and means. Information security management systems. General overview and terminology. Moscow: FGUP STANDARTINFORM. 2014. [in Russian]

49. Mayzaud A., Badonnel R., Chrisment I. A Taxonomy of Attacks in RPL-based Internet of Things // International Journal of Network Security. 2016. Vol. 18. No. 3. P. 459-473.

50. Zhu B., Joseph A., Sastry S. A taxonomy of cyber attacks on SCADA systems // 2011 International conference on internet of things and 4th international conference on cyber, physical and social computing. IEEE, 2011. P. 380-388.

51. Humayed A., Lin J., Li F. and Luo B. Cyber-physical systems security — A survey // IEEE Internet of Things Journal. 2017. Vol. 4. No. 6. P. 1802-1831.

52. Alguliyev R., Imamverdiyev Y., Sukhostat L. Cyber-physical systems and their security issues // Computers in Industry. 2018. Vol. 100. P. 212-223.

53. Ashibani Y., Mahmoud Q. Cyber physical systems security: Analysis, challenges and solutions // Computers & Security. 2017. Vol. 68. P. 81-97.

54. Gao Y., Peng Y., Xie F., Zhao W., Wang D., Han X., Lu T. and Li Z. Analysis of security threats and vulnerability for cyber-physical systems // Proceedings of 2013 3rd International Conference on Computer Science and Network Technology. IEEE, 2013. P. 50-55.

55. Makhdoom I., Abolhasan M., Lipman J., Liu R. and Ni W. Anatomy of threats to the internet of things // IEEE communications surveys & tutorials. 2018. Vol. 21. No. 2. P. 1636-1675.

56. Yampolskiy M., Horváth P., Koutsoukos X., Xue Y. and Sztipanovits J. A language for describing attacks on cyber-physical systems // International Journal of Critical Infrastructure Protection. 2015. Vol. 8. P. 40-52.

57. Heartfield R., Loukas G., Budimir S., Bezemskij A., Fontaine J., Filippoupolitis A. and Roesch E. A taxonomy of cyber-physical threats and impact in the smart home // Computers & Security. 2018. Vol. 78. P. 398-428.

58. Alekseev D., Ivanenko K., Ubirailo V. Classification of threats to information security. Simvol nauki — Science symbol. 2016. Vol. 9-1. P. 18–20. [in Russian]

59. Ashibani Y., Mahmoud Q. Cyber physical systems security: Analysis, challenges and solutions // Computers & Security. 2017. Vol. 68. P. 81-97.

60. Desnitsky V., Levshun D., Chechulin A. and Kotenko I. Design Technique for Secure Embedded Devices: Application for Creation of Integrated Cyber-Physical Security System // J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl. 2016. Vol. 7. No. 2. P. 60-80.

61. Kotenko I., Levshun D., Chechulin A., Ushakov I. and Krasov A. Integrated approach to provide security of cyber-physical systems based on microcontrollers. Voprosy Kiberbezopasnosti. 2018. Vol. 3(27). P. 29-38. [in Russian]

62. Zegzhda D., Vasilev U., Poltavtseva M., Kefele I., Borovkov A. Advanced production technologies security in the era of digital transformation. Voprosy Kiberbezopasnosti. 2018. Vol. 2(26). P. 2–14. [in Russian]

63. Frahim J. Securing the Internet of Things: A Proposed Framework. Cisco White Paper,March 2015.

64. Gaifulina D.A. Analytical review of methods for detecting network layer anomalies in cyber-physical systems. Al’manah nauchnyh rabot molodyh uchenyh Universiteta ITMO — Almanac of scientific works of young scientists of ITMO University. 2018. Issue 1. P. 4–5. [in Russian].

65. Kotenko I., Doynikova E. Vulnerabilities assessment techniques: use for the computer systems security analysis. Zashchita informacii. Insajd — Information Security. Inside. 2011. Vol. 4. P. 74–81. [in Russian]

66. Desmit Z., Elhabashy A., Wells L., Camelio J. An approach to cyber-physical vulnerability assessment for intelligent manufacturing systems. Journal of Manufacturing Systems. 2017. Vol. 43. P. 339–351.

67. Radanliev P., De Roure D., Nicolescu R., Huth M., Montalvo M., Cannady S. and Bumap P. Future developments in cyber risk assessment for the internet of things // Computers in industry. 2018. Vol. 102. P. 14-22.

68. Lyu X., Ding Y., Yang S. Safety and security risk assessment in cyber-physical systems // IET Cyber-Physical Systems: Theory & Applications. 2019. Vol. 4. No. 3. P. 221-232.

69. Telegina M., Yannikov I., Kudelkin V., Ushakov I. Models and methods for safety assessment of potentially dangerous objects. Intellektual’nye sistemy v

proizvodstve — Intelligent systems in production. 2017. Issue 15. Vol. 1. P. 118–121. [in Russian]

70. Kulagina I., Iskhakova A., Galin R. Modeling the practice of aggression in the socio-cyber-physical environment. Vestnik Tomskogo gosudarstvennogo universiteta. Filosofiya. Sotsiologiya. Politologiya — Tomsk State University Journal of Philosophy, Sociology and Political Science. 2019. Vol. 52. P. 147–161. [in Russian]

71. Garate V. Analysis of the security level of corporate networks in the context of social engineering attacks. Izvestiya SPbGETU «LETI» – Izvestiya ETU LETI. 2017. Issue 3. P. 12–15. [in Russian]

72. Hu F., Lu Y., Vasilakos A., Hao Q., Ma R., Patil Y., Zhang T., Lu J., Li X. and Xiong N. Robust cyber–physical systems: Concept, models, and implementation // Future generation computer systems. 2016. Vol. 56. P. 449-475.

73. Sirjani M. Analysing Real-time Distributed Systems using Timed Actors // 23rd IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT). IEEE, 2019. P. 1-1.

74. Dai W., Pang C., Vyatkin V., Christensen J. and Guan X. Discrete-event-based deterministic execution semantics with timestamps for industrial cyber-physical systems // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2017. Vol. 50. No. 3. P. 851-862.

75. Srivastava A., Morris T., Ernster T., Vellaithurai C., Pan S. and Adhikari U. Modeling cyber-physical vulnerability of the smart grid with incomplete information // IEEE Transactions on Smart Grid. 2013. Vol. 4. No. 1. P. 235-244.

76. Xinyu C., Huiqun Y., Xin X. Verification of Hybrid Chi model for cyber-physical systems using PHAVer // 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. IEEE, 2013. P. 122-128.

77. Nuzzo P., Sangiovanni-Vincentelli A., Bresolin D., Geretti L. and Villa T. A platform-based design methodology with contracts and related tools for the design of cyber-physical systems // Proceedings of the IEEE. 2015. Vol. 103. No. 11. P. 2104-2132.

78. Di Nitto E., Matthews P., Petcu D. and Solberg A. Model-driven development and operation of multi-cloud applications: the MODAClouds approach. Springer Nature, 2017.

79. Iannucci S., Abdelwahed S., Montemaggio A., Hannis M., Leonard L., King J. and Hamilton J. A model-integrated approach to designing self-protecting systems // IEEE Transactions on Software Engineering. 2018. Vol. 46. No. 12. P. 1380-1392.

80. Karagiannis D., Mayr H., Mylopoulos J. Domain-specific conceptual modeling. Springer International Publishing, 2016.

81. Rahman M., Mahmud M. and Pota H. Multi-agent approach for enhancing security of protection schemes in cyber-physical energy systems // IEEE transactions on industrial informatics. 2016. Vol. 13. No. 2. P. 436-447.

82. Balasubramaniyan S., Srinivasan S., Buonopane F., Subathra B., Vain J. and Ramaswamy S. Design and verification of Cyber-Physical Systems using TrueTime, evolutionary optimization and UPPAAL // Microprocessors and microsystems. 2016. Vol. 42. P. 37-48.

83. David A., Larsen K., Legay A., Mikučionis M. and Poulsen D. Uppaal SMC tutorial // International Journal on Software Tools for Technology Transfer. 2015. Vol. 17. No. 4. P. 397-415.

84. Penas O., Plateaux R., Patalano S. and Hammadi M. Multi-scale approach from mechatronic to Cyber-Physical Systems for the design of manufacturing systems // Computers in Industry. 2017. Vol. 86. P. 52-69.

85. Friedenthal S., Moore A., Steiner R. A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014.

86. H. Elmqvist. Dymola — a structured modeling language for large continuous systems. Department of Automatic Control, Lund Institute of Technology (LTH), 2018.

87. Elmqvist H., Otter M. Innovations for future Modelica // Proceedings of 12th International Modelica Conference. Linköping University Electronic Press, 2017.

88. Wymore A. W. Model-based systems engineering. CRC press, 2018. Vol. 3.

89. Fritzson P. Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach. John Wiley & Sons, 2014.

90. Chaturvedi D. Modeling and simulation of systems using MATLAB and Simulink. CRC press, 2017.

91. Fajstrup L., Goubault E., Haucourt E., Mimram S. and Raussen M. Directed algebraic topology and concurrency. Berlin : Springer, 2016. Vol. 138.

92. Shen W., Wang J., Luo P. and Wang M. A graph-based approach for ontology population with named entities // Proceedings of the 21st ACM international conference on Information and knowledge management. 2012. P. 345-354.

93. Zhu B., Roy U. Modeling and validation of a web ontology language based disassembly planning information model // Journal of Computing and Information Science in Engineering. 2018. Vol. 18. No. 2.

94. Blouin D., Borde E. AADL: A Language to Specify the Architecture of Cyber-Physical Systems // Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems. Springer, Cham, 2020. P. 209-258.

95. Seiger R., Keller C., Niebling F. and Schlegel T. Modelling complex and flexible processes for smart cyber-physical environments // Journal of Computational Science. 2015. Vol. 10. P. 137-148.

96. Nassar N., Kosiol J., Kehrer T. and Taentzer G. Generating Large EMF Models Efficiently // International Conference on Fundamental Approaches to Software Engineering. Springer, Cham, 2020. P. 224-244.

97. Cremers C. Symbolic security analysis using the Tamarin prover // 2017 Formal Methods in Computer Aided Design (FMCAD). IEEE, 2017. P. 5-5.

98. Srinivasan S., Buonopane F., Vain J. and Ramaswamy S. Model checking response times in Networked Automation Systems using jitter bounds //Computers in Industry. 2015. Vol. 74. P. 186-200.

99. Blanchet B. Automatic verification of security protocols in the symbolic model: The verifier proverif // Foundations of security analysis and design VII. Springer, Cham, 2013. P. 54-87.

100. Chothia T., Smyth B., Staite C. Automatically checking commitment protocols in proverif without false attacks // International Conference on Principles of Security and Trust. Springer, Berlin, Heidelberg, 2015. P. 137-155.

101. Chadha R., Cheval V., Ciobâcă Ş. and Kremer S. Automated verification of equivalence properties of cryptographic protocols // ACM Transactions on Computational Logic (TOCL). 2016. Vol. 17. No. 4. P. 1-32.

102. Avanesov T., Chevalier Y., Rusinowitch M. and Turuani M. Intruder deductibility constraints with negation. Decidability and application to secured service compositions // Journal of Symbolic Computation. 2017. Vol. 80. P. 4-26.

103. [Armando A., Arsac W., Avanesov T., Barletta M., Calvi A., Cappai A., Carbone R., Chevalier Y., Compagna L., Cuéllar J., Erzse G., Frau S., Minea M., Mödersheim S., Oheimb D., Pellegrino G., Ponta S., Rocchetto M., Rusinowitch M., Dashti M., Turuani M., Viganò L. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures //International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2012. P. 267-282.

104. Levshun D., Chevalier Y., Kotenko I. and Chechulin A. Design and verification of a mobile robot based on the integrated model of cyber-Physical systems // Simulation Modelling Practice and Theory. 2020. Vol. 105. P. 102151.

105. Faily S. Further Applications of CAIRIS for Usable and Secure Software Design // Designing Usable and Secure Software with IRIS and CAIRIS. Springer, Cham, 2018. P. 239-254.

106. Kobashi T., Washizaki H., Yoshioka N., Kaiya H., Okubo T. and Fukazawa Y. Designing secure software by testing application of security patterns // Exploring Security in Software Architecture and Design. IGI Global, 2019. P. 136-169.

107. Ardeshiricham A., Hu W., Marxen J. and Kastner R. Register transfer level information flow tracking for provably secure hardware design // Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017. P. 1691-1696.

108. Zhang D., Wang Y., Suh G. and Myers A. A hardware design language for timing-sensitive information-flow security // Acm Sigplan Notices. 2015. Vol. 50. No. 4. P. 503-516.

109. Xu X., He B., Yang W., Zhou X. and Cai Y. Secure transmission design for cognitive radio networks with poisson distributed eavesdroppers // IEEE Transactions on Information Forensics and Security. 2015. Vol. 11. No. 2. P. 373-387.

110. Wang B., Zhong S., Dong X. On the novel chaotic secure communication scheme design // Communications in Nonlinear Science and Numerical Simulation. 2016. Vol. 39. P. 108-117.

111. Takahashi S., Ikeda T., Shinagawa Y., Kunii T. and Ueda M. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data // Computer Graphics Forum. Edinburgh, UK: Blackwell Science Ltd, 1995. Vol. 14. No. 3. P. 181-192.

112. Saleem K., Derhab A., Al-Muhtadi J. and Shahzad B. Human-oriented design of secure Machine-to-Machine communication system for e-Healthcare society // Computers in Human Behavior. 2015. Vol. 51. P. 977-985.

113. Huang J. and Huang C. Design and verification of secure mutual authentication protocols for mobile multihop relay WiMAX networks against rogue base/relay stations // Journal of Electrical and Computer Engineering. 2016.

114. Wang Z., Karpovsky M., Bu L. Design of reliable and secure devices realizing Shamir's secret sharing // IEEE Transactions on Computers. 2015. Vol. 65. No. 8. P. 2443-2455.

115. Scott-Hayward S. Design and deployment of secure, robust, and resilient SDN Controllers // Proceedings of the 2015 1st IEEE conference on network Softwarization (NetSoft). IEEE, 2015. P. 1-5.

116. Lin Z., Yu S., Lü J., Cai S. and Chen G. Design and ARM-embedded implementation of a chaotic map-based real-time secure video communication system // IEEE Transactions on circuits and systems for video technology. 2014. Vol. 25. No. 7. P. 1203-1216.

117. Official website of Google Internet of Things Cloud solution. URL: https://cloud.google.com/solutions/iot/. Online: accessed 16.06.2021.

118. Official website of Google Internet of Things DeviceSDK solution. URL: https://bit.ly/2K6rbGQ. Online: accessed 16.06.2021.

119. Official website of ARM Platform Security Architecture solution. URL: https://www.arm.com/why-arm/architecture/platform-security-architecture. Online: accessed 16.06.2021.

120. Official website of Kaspersky Industrial Cyber-Security solution. URL: https://ics.kaspersky.com/. Online: accessed 16.06.2021.

121. Official website of Kaspersky Operation System. URL: https://os.kaspersky.com/. Online: accessed 16.06.2021.

122. Official website of Microsoft Azure Internet of Things solution. URL: https://azure.microsoft.com/en-us/overview/iot/. Online: accessed 16.06.2021.

123. Official website of Microsoft Security Development Lifecycle with examples and documentation. URL: https://www.microsoft.com/enus/securityengineering/sdl/. Online: accessed 16.06.2021.

124. Official website of Intel Internet of Things Platform. URL: https://www.intel.com/content/www/us/en/internet-of-things/iot-platform.html. Online: accessed 16.06.2021.

125. Official website of MindSphere — cloud-based, open IoT operating system from Siemens. URL: https://new.siemens.com/global/en/products/software/mindsphere.html. Online: accessed 16.06.2021.

126. Official website of Cisco Secure Development Lifecycle. URL: https://www.cisco.com/c/dam/en_us/about/doing_business/trust-center/docs/cisco-secure-development-lifecycle.pdf. Online: accessed 16.06.2021.

127. Desnitsky V., Chechulin A., Kotenko I., Levshun D. and Kolomeec M. Application of a technique for secure embedded device design based on combining security components for creation of a perimeter protection system // 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). IEEE, 2016. P. 609-616.

128. Desnitsky V., Kotenko I., Chechulin A. Configuration-based approach to embedded device security // International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, Berlin, Heidelberg, 2012. P. 270-285.

129. Official website of SecFutur project — Design of Secure and energy-efficient embedded systems for Future internet applications. URL: https://cordis.europa.eu/project/id/256668. Online: accessed 16.06.2021.

130. Chechulin A., Kotenko I., Desnitsky V. An approach for network information flow analysis for systems of embedded components // International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, Berlin, Heidelberg, 2012. P. 146-155.

131. Riedmüller S., Brecht U., Sikora A. IPsec for Embedded Systems // ITCS 2005. 2005. P. 8.

132. Chelli K. Security issues in wireless sensor networks: Attacks and countermeasures // Proceedings of the world congress on engineering. 2015. Vol. 1. No. 20. P. 876-3423.

133. Levshun D., Chechulin A., Kotenko I. A technique for design of secure data transfer environment: Application for I2C protocol // 2018 IEEE Industrial Cyber-Physical Systems (ICPS). IEEE, 2018. P. 789-794.

134. Feilner M. OpenVPN: Building and integrating virtual private networks. Packt Publishing Ltd, 2006.

135. Official website of the Python Programming Language. URL: https://www.python.org/. Online: accessed 16.06.2021.

136. Official website of PostgreSQL — the powerful, open-source object-relational database system. URL: https://www.postgresql.org/. Online: accessed 16.06.2021.

137. Official website of psycopg — PostgreSQL database adapter for the Python programming language. URL: https://pypi.org/project/psycopg2/. Online: accessed 16.06.2021.

138. Official website of the tkinter package — the standard Python interface to the Tk GUI toolkit. URL: https://docs.python.org/3/library/tkinter.html. Online: accessed 16.06.2021.

139. Official website of the pygubu tool — a RAD tool to enable quick & easy development of user interfaces for the python tkinter module.. URL: https://pypi.org/project/pygubu/. Online: accessed 16.06.2021.

140. Official website of networkx — a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. URL: https://networkx.org/. Online: accessed 16.06.2021.

141. Official website of the json package that represents Python dictionaries in JSON format. URL: https://docs.python.org/3/library/json.html. Online: accessed 16.06.2021.

142. Official website of the functools module of Python that allows the use and extension of callable objects without completely rewriting them. URL: https://docs.python.org/3/library/functools.html. Online: accessed 16.06.2021.

143. Official website of the time module of Python that provides various time-related functions. URL: https://docs.python.org/3/library/time.html. Online: accessed 16.06.2021.

144. Official website of PL/pgSQL — SQL Procedural Language for PostgreSQL databases. URL: postgresql.org/docs/13/plpgsql-statements.html. Online: accessed 16.06.2021.

145. Balzarotti D., Monga M., Sicari S. Assessing the risk of using vulnerable components // Quality of protection: security measurements and metrics, Advances in Information Security 23. Springer, New York, 2006. P.65-77.

146. Runeev A., Kotenko I. Fundamentals of control theory in military systems: a tutorial. Part 2. 2000. 158 p.

147. Dmitry Levshun. Algorithm for the formation of the component composition of a secure microcontroller-based system // Proceedings of the XII Saint-Petersburg Interregional conference Information security of regions of Russia (IBRR-2021). St. Petersburg, Russia, 2021. P. 88-90. [in Russian]

148. Daniel Zelle, Roland Rieke, Christian Plappert, Christoph Kraub, Dmitry Levshun, Andrey Chechulin. SEPAD – Security Evaluation Platform for Autonomous Driving // Proceedings of the 28th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP-2020). Vesteos, Sweden, March 11-13, 2020. P. 413-420. DOI: 10.1109/PDP50117.2020.00070.

149. Dmitry Levshun. Requirements for the methodology for design and verification of secure cyber-physical systems // Proceeding of the XVII St. Petersburg International Conference Regional Informatics (RI-2020). St. Petersburg, Russia, 2020. P. 144-146. [in Russian]

150. Dmitry Levshun. Approach to the formation of requirements in the design process of secure cyber-physical systems // Proceedings of the IV Interregional Scientific-Practical Conference Advanced National Information Systems and Technologies (ANIST-2020). Sevastopol, Russia, 2020. P. 239-240. [in Russian]

151. Dmitry Levshun. Approach to the formation of specifications for secure cyber-physical systems // Proceedings of the IV Interregional Scientific-Practical Conference Advanced National Information Systems and Technologies (ANIST-2020). Sevastopol, Russia, 2020. P. 239-240. [in Russian]

152. Dmitry Levshun. An attacker model for a modern cyber-physical system // Proceedings of the IX International Scientific, Technical and Scientific Methodological Conference Actual Problems of Information Telecommunications in Science and Education. St. Petersburg, Russia, 2020. P. 679-682. [in Russian]

153. Dmitry Levshun, Andrey Chechulin. The aspects of the verification of secure cyber-physical systems // Proceedings of the XI Saint-Petersburg Interregional conference “Information security of regions of Russia” (IBRR-2019). St. Petersburg, Russia, 2019. P. 133-134. [in Russian]

154. Dmitry Levshun. Application of modeling for verification of cyber-physical systems security // Proceedings of the XI Saint-Petersburg Interregional conference “Information security of regions of Russia” (IBRR-2019). St. Petersburg, Russia, 2019. P. 131-132. [in Russian]

155. Dmitry Levshun, Igor Kotenko, Andrey Chechulin. The Integrated Model of Secure Cyber-Physical Systems for their Design and Verification // Proceedings of the 13th International Symposium on Intelligent Distributed Computing (IDC 2019). St. Petersburg, Russia, 2019. Studies in Computational Intelligence. Vol. 868. 2020. P.333-343. DOI: 10.1007/978-3-030-32258-8_39.

156. Dmitry Levshun, Yannick Chevalier, Igor Kotenko, Andrey Chechulin. Secure Communication in Cyber-Physical Systems // Proceedings of the the 3rd International Symposium on Mobile Internet Security (MobiSec 2018). Cebu, Philippines, 2018. 9 p. URL: isyou.info/conf/mobisec18/mobisec18-book-ver1.pdf.

157. Dmitry Levshun. The approach to design of secure systems based on embedded devices // Proceedings of the X St. Petersburg Interregional Conference "Information Security of the regions of Russia" (IBRR-2017). St. Petersburg, Russia, 2017. Issue 4. P. 414-416. [in Russian]

158. Dmitry Levshun, Andrey Chechulin, Igor Kotenko. Design Lifecycle for Secure Cyber-Physical Systems based on Embedded Devices // Proceedings of the 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2017). Bucharest, Romania, 2017. P. 277-282. DOI: 10.1109/IDAACS.2017.8095090.

159. Dmitry Levshun, Andrey Chechulin, Igor Kotenko. Design of secure microcontroller-based systems: application to mobile robots for perimeter monitoring // Sensors. 2021. Accepted 08.12.2021.

160. Dmitry Levshun, Igor Kotenko, Andrey Chechulin. The application of the methodology for secure cyber–physical systems design to improve the semi-natural model of the railway infrastructure // Microprocessors and Microsystems. Vol. 87. 2021. P. 103482. DOI: 10.1016/j.micpro.2020.103482.

161. Dmitry Levshun, Diana Gaifulina, Andrey Chechulin and Igor Kotenko. Problematic Issues of Information Security of Cyber-Physical Systems // Informatics and Automation. Vol. 19. No. 5. 2020. P. 1050-1088. ISSN 2078-9181 (2078-9599). DOI: 10.15622/ia.2020.19.5.6. [in Russian]

162. Dmitry Levshun, Igor Kotenko. Application for the design of secure microcontroller-based physical security systems. Federal Service for Intellectual

Property. Certificate #2021680236. Registered in the Computer Program Registry 08.12.2021.

163. Dmitry Levshun. Database for the design of secure microcontroller-based physical security systems. Federal Service for Intellectual Property. Certificate #2021622496. Registered in the Computer Program Registry 15.11.2021.

164. Dmitry Levshun. Component of traffic generation for cyber-physical systems based on I2C protocol. Federal Service for Intellectual Property. Certificate #2018664325. Registered in the Computer Program Registry 14.11.2018. URL:

165. Dmitry Levshun, Igor Kotenko, Andrey Chechulin. Repository for heterogeneous data from the hardware elements of the smart home. Federal Service for Intellectual Property. Certificate #2017620996. Registered in the Database Registry 01.09.2017.

166. Dmitry Levshun, Andrey Chechulin, Igor Kotenko. System for support and management of database of the room access control and management system based on the contactless smart cards. Federal Service for Intellectual Property. Certificate #2016612543. Registered in the Computer Program Registry 01.03.2016.

167. Dmitry Levshun, Andrey Chechulin. Database of the logging server of a secure access control system for Smart House model. Federal Service for Intellectual Property. Certificate #2016621608. Registered in the Database Registry 29.11.2016.

Appendix A. Modeling of the perimeter monitoring system

This appendix describes the process of modeling of microcontroller-based devices of the physical security system, described in Section 6.1.1, namely, the server, charging stations and mobile robots, in accordance with the extendable set-based hierarchical relational model, presented in Chapter 3.

In terms of the extendable set-based hierarchical relational model it means that mbs from Section 6.1.1 does not contain any sub-systems and has the following three types of building blocks: a server, a set of mobile robots and a set of charging stations. It means that BB of mbs can be represented as follows:

$$BB = (bb_1, BB_2, BB_3),$$

where bb_1 — the server of mbs; BB_2— set of mobile robots $bb_{2i} \in BB_2 \mid i \in 1 \dots n$ of mbs; BB_3 — set of charging stations $bb_{3i} \in BB_3 \mid i \in 1 \dots m$ of mbs.

Mobile robots and charging stations are communicating with the server via Wi-Fi connection, where the server is the host of the access point. Additionally, stations are communicating with robots via Infrared (IR) connection during their parking for charging, while the charging process is wireless and done via Qi communication. It means that L_{mbs}can be represented as follows:

$$L_{mbs} = (L_{mbs_1}, L_{mbs_2}, L_{mbs_3}),$$

where L_{mbs_1} — Wi-Fi communication between bb_1(server) and BB_2(mobile robots) as well as between bb_1 and BB_3(charging stations) of mbs; L_{mbs_2} — IR communication between BB_2 and BB_3; L_{mbs_3} — Qi communication between BB_2 and BB_3.

As was mentioned in Chapter 3, properties of the mbs are represented as follows:

$$p = (FR, NL, PF, PR),$$

where FR – a set of functional requirements; NL – a set of non-functional limitations; PF – a set of provided functionalities; PR – a set of provided resources.

Let's consider them in more detail.

FR represents needs of mbs — functionality that satisfaction is necessary for it to be able to work, see Table A.1.

Table A.1. Functional requirements of the designed system

		Description
FR	fr_1	power source for charging stations
	fr_2	power source for server
	fr_3	secure connection of server to update system

NL represents needs of mbs — limitations which satisfaction is necessary for it to be able to work, see Table A.2.

Table A.2. Non-functional limitations of the designed system

		Description
NL	nl_1	space for charging stations placement
	nl_2	space for server placement
	nl_3	space for mobile robots' movement
	nl_4	the environment does not contain elements that can lead to incorrect operation of device sensors
	nl_5	the environment does not contain elements that can lead to incorrect operation of wireless communications

PF represents capabilities of mbs — functionality that it can provide, see Table A.3.

Table A.3. Provided functionality of the designed system

		description
PF	pf_1	to monitor the perimeter
	pf_2	to add new mobile robots
	pf_3	to add new charging stations
	pf_4	to monitor locations of mobile robots and charging stations
	pf_5	to monitor occupancy of charging stations
	pf_6	to monitor charge state of mobile robots
	pf_7	to update software and firmware of system devices
	pf_8	to be secure against attackers with $ac = 4,\ kn = 2,\ rs = 2$

PR represents capabilities of mbs — resources that it can provide, see Table A.4.

Table A.4. Provided resources of the designed system

		description
PR	pr_1	to store system data in relational database
	pr_2	to run executable applications compatible with Linux operating systems

Note that Influence of AA can be modeled through canceling or reducing of some of PF and PR as well as through enhancing or introducing of FR and NL. For example, based on pf_8 system must be secure against iws — interception, modification or termination of wireless communications. To prevent such a class of attack actions, the following security elements must be the part of the designed system: strong encryption mechanism on access point, strong login credentials and public key pair-based authentication.

Thus, mbs of mobile robots for perimeter monitoring can be represented as follows:

$$
\begin{aligned}
&mbs = (BB,\ L_{mbs},\ a,\ AA,\ p_{mbs}),\\
&BB = (bb_1,\ BB_2,\ BB_3),\\
&L_{mbs} = (L_1, L_2, L_3),\\
&a = (ac,\ kn,\ rs) \mid ac = 4,\ kn = 2,\ rs = 2,\\
&AA = \{(cl_1,\ oj_1, sj_1),\ \ldots,\ (cl_n,\ oj_n, sj_n)\} \mid n \in N,\\
&p_{mbs} = (FR\ ,\ NL\ ,\ PF\ ,\ PR\ )
\end{aligned}
$$

where bb_1 — the server; BB_2 — set of mobile robots $bb_{2i} \in BB_2 \mid i \in 1\ldots n$; BB_3 — set of charging stations $bb_{3i} \in BB_3 \mid i \in 1\ldots m$; L_1 — Wi-Fi communication between bb_1 and BB_2 as well as between bb_1 and BB_3; L_2 — IR communication between BB_2 and BB_3 ; L_3 — Qi communication between BB_2 and BB_3 of mbs; ac — type of access a has to mbs; kn — type of knowledge a has about mbs; rs — type of resources available to a to compromise mbs; cl_i — *i*-th class of attack; oj_i — *i*-th object of attack, helps to link aa_i with the target element(s); sj_i — *i*-th subject of attack, helps to link aa_i with a that is capable enough for its successful realization; FR — set of functional requirements of mbs; NL — set of non-functional limitations of mbs; PF — set of provided functionalities of mbs; PR — set of provided resources of mbs.

It means that L_{mbs_1} — Wi-Fi communications between the server of the designed system and mobile robots/charging stations — can be represented as follows:

$$L_{mbs_1} = (IEEE\ 800.11,\ wireless\ 2.4\ GHz,\ \{bb_1 \leftrightarrow BB_2,\ bb_1 \leftrightarrow BB_3\},\ p_{L_{mbs_1}}),$$

where bb_1 — the server of mbs; BB_2 — set of mobile robots of mbs; BB_3 — set of charging stations of mbs; $p_{L_{mbs_1}}$ — properties of L_{mbs_1}.

Let us consider properties of L_{mbs_1} in more detail.

FR represents the functionality that satisfaction is necessary for L_{mbs_1} to be able to work. It contains:

- fr_1 — physical connection to Wi-Fi module with correct voltage;
- fr_2 — software library for work with Wi-Fi modules.

NL represents the limitations which satisfaction is necessary for L_{cps_1} to be able to work. It contains:

- nl_1 — all communication parties are supporting the same Wi-Fi standards;
- nl_2 — the environment does not contain elements that can lead to incorrect operation of 2.4 GHz wireless communications.

PF represents the functionality that L_{mbs_1} can provide. It contains:

- pf_1 — wireless communication between devices and the access point;
- pf_2 — connected devices are sharing throughput;
- pf_3 — WPA2-PSK security.

PR represents the resources that L_{mbs_1} can provide. It contains:

- pr_1 — to create access points;
- pr_2 — to connect devices to access points.

Note that properties of L_{mbs_1} are connected with properties of the designed microcontroller-based system. Other links between elements of the mbs can be represented in the same way. Thus, all elements of the developed extendable set-based hierarchical relational model were presented in this chapter.

A1. Modeling of the server

The server of the designed system is based on Raspberry Pi single board computer with Raspbian OS. This operation system should be installed on a micro-SD card with not less than 16 GB space. It could be done via Imager — a special software tool. After that, it is required to slot this micro-SD card into the single board computer, so it would be able to identify the operating system and use micro-SD card memory space as its own. When it is done, it becomes possible to install PostgreSQL database server, Python compiler as well as different drivers and libraries that are necessary for the server to perform its functionality. Moreover, it becomes possible to configure the server's operating system, database access, software update policy and Wi-Fi access point. To provide a power supply, the server is connected to a power bank which in its turn should be connected to the home power supply.

Thus, in terms of the developed model, bb_1 contains:

- sub-block $bb_1^{'}$: Raspberry Pi single board computer (represents building block that combines hardware and software elements) in which another building block is slotted — a micro-SD card (hardware element) with pre-installed Raspbian OS (software element);
- hardware element hw_1: power bank that provides power supply to $bb_1^{'}$ and connected to home power grid;
- software element sw_1: application with graphical user interface and database that grants a possibility to monitor events of the system, state and location of mobile robots and charging stations;
- software element sw_2: configured Wi-Fi access point with strong login credentials and encryption;
- software element sw_3: configured update system for keeping bb_1 software up-to-date with the connection to the remote server.

As was mentioned before, sub-block $bb_1^{'}$ is representing a combination of two other building blocks that are connected through putting an SD card into a special slot on the single board computer as well as due to correct formatting of the SD card and the work of the drivers. On the basis of $bb_1^{'}$ it becomes possible to install and configure additional software elements. Moreover, sw_1 represents an application that contains a combination of three other software elements – database, script, graphical user interface – that were connected via special drivers and libraries during the compilation process. It means that L_{bb_1} can be represented as follows:

$$L_{bb_1} = (L_1, L_2, L_3),$$

where L_1 – the USB to micro-USB connection between $bb_1^{'}$ and hw_1 of bb_1; L_2 – libraries and drivers that provide interaction between $bb_1^{'}$ and $sw_1 - sw_3$; L_3 – the Internet connection between sw_3 and remote update server.

As for properties of bb_1 – the server of the system – it is important to note that they are connected with the properties of mbs and partially represent them.

FR represents the needs of bb_1 – functionality that satisfaction is necessary for the server of the system to be able to work. It contains:

- fr_1 – power source;
- fr_2 – secure connection between the update system and remote server.

NL represents the needs of bb_1 – limitations which satisfaction is necessary for the server of the system to be able to work. It contains:

- nl_1 – space for the device;
- nl_2 – the environment does not contain elements that can lead to incorrect operation of wireless communications.

PF represents the capabilities of bb_1 – functionality that the server of the system can provide. It contains:

- pf_1 – to add new mobile robots;
- pf_2 – to add new charging stations;
- pf_3 – to update software, drivers and libraries;
- pf_4 – to communicate with mobile robots and charging stations;
- pf_5 – to be secure against attackers with $ac = 4, kn = 2, rs = 2$.

PR represents the capabilities of bb_1 – resources that the server of the system can provide. It contains:

- pr_1 – to store system data in relational database;
- pr_2 – to run executable applications compatible with Linux operating systems.

Based on the requirements, the algorithm can form the device that represents the server of the system step-by-step. For such a device it is required to have:

- one-board computer with in-build Wi-Fi and Ethernet interfaces that supports high-level operating systems with graphical user interface, has not less than 1 GB RAM and supports disk space extension based on micro-SD cards;

- micro-SD card with not less than 16 GB disk space that is supported by the selected one-board computer;
- 32-bit operating system image that can be installed on micro-SD that is supported by the selected one-board computer;
- database that supports SQL queries which structure is sufficient enough to store system data and events;
- application with graphical user interface and connection to the selected database that presents processed data to the operator of the system and communicates with other devices of the system;
- battery that can power up the selected one-board computer as well as be connected to the home power grid.

But this is without taking into account security requirements. So let us consider them in more detail. The ability to be secure against attackers with $ac = 4, kn = 2, rs = 2$ can be interpreted as to be secure against: rpt — replacement of the electronic component; rmt — removement of the electronic component; imw — interception, modification or termination of wired communications; iec — increased energy consumption; iws — interception, modification or termination of wireless communications; soc — social engineering; pwr — power failure; web — disruption of web services; dbd — database disruption.

To prevent rpt and rmt on the level of the server a vandal-proof device case (hardware element) should be used for the server of the system.

To prevent pwr on the level of the server in addition to a vandal-proof device case, the selected power bank capacity should be big enough to provide power supply to the selected one-board computer in the absence of power supply from the home power grid for 4 hours.

To prevent imw and web on the level of the server no additional security elements are required because there is no wired communications in the device as well as no web-services.

To prevent iec and iws on the level of the server additional software elements are required: an algorithm for behavior-based anomaly detection and a secure configuration of the wireless access point (strong encryption, strong login credentials, public key pair-based authentication).

To prevent dbd on the level of the server it is required to add the following software elements to the device: algorithm for the validation of the input data (an additional requirement for the selected application), configuration with strict database access policy (an additional requirement for the selected database), configuration with strong login credentials (an additional requirement for the selected database),

configuration with separate database users for different operations (an additional requirement for the selected database).

To prevent soc on the level of the server it is required to train operators and users of the system. This requirement cannot be translated into software or hardware elements of the device, so would be transferred to the stakeholder as an additional recommendation during the system implementation.

Thus, the step-by-step formation of the server of the system:

1. Combination of a *micro-SD card* and an *operating system image* with the help of additional software tools (for example Imager can be used with Raspberry Pi one board computers to install Raspbian).
2. Combination of the *micro-SD card with installed operating system* and a *single-board computer* with help of micro-SD slot.
3. Combination of the *single-board computer with micro-SD card* and a *power bank* with the help of the USB connection, while the power bank is connected to the home power grid. At this step the algorithm formed a working one-board computer that will require additional configuration.
4. Secure configuration of the *wireless access point* on the *powered up single-board computer*. Such a configuration requires the configuration based on at least WPA2-PSK to cover requirements related to the encryption algorithm, login credentials and authentication.
5. Combination of the *single-board computer with wireless access point* and a *database server* (for example, with help of sudo apt update/install in Linux-based systems). Secure configuration of the database server: only localhost connections, strong login credentials. Creation of the database to store system data and events and its secure configuration: separate users for different operations, strict access policy, strong login credentials.
6. Combination of the *single-board computer with database* and an *executable application with graphical user interface* based on the installation of required libraries, drivers and compilers (not only to work in the environment of the selected operating system, but also to work with the selected database). Extension of the application functionality with the validation of received data before it is written into the selected database. Extension of the application functionality with an algorithm for behavior-based anomaly detection that is trained to detect resource depletion attacks.
7. Secure configuration of the *Ethernet connection* to the *remote update server* on the *single-board computer with application*. This update server is required to provide software updates for the server as well as firmware updates for mobile robots and charging stations.
8. Combination of the *single-board computer* that represents the server of the designed system and a *vandal-proof device case* to provide additional security from physical attacks.

At this point the server of the system is formed based on the abstract hardware and software elements as well as building blocks. To make it more concrete, suitable real-world examples should be used.

For example, as the selected single-board computer it is possible to use Raspberry Pi 3, 3 B+, 4 and 4 B. And if the Raspberry Pi platform is selected, as an operating system it is possible to install Raspbian, Ubuntu MATE, Pidora, Linutop, SARPi and other systems, while the application can be written in Python, Java, C++ and so on. As for the database server, MySQL or PostgreSQL are the most popular ones.

One of possible alternatives of the server is presented in Table A.5.

Table A.5. Component composition of the server

Abstract component	Selected physical component
single-board computer	Raspberry Pi 4 Model B with Broadcom BCM2711, ARM Cortex A72 1.5 GHz, 2 GB SDRAM, Ethernet, Wi-Fi, USD, HDMI, up to 1280 mAh
micro-SD card	Samsung Pro Endurance 32 GB Class 10
operating system	Raspberry Pi OS with desktop and recommended software, 2863 MB. Configuration of the secure Wi-Fi access point (based on WPA2 PSK for the strong encryption algorithm and login credentials as well as public key pair-based authentication). Configuration of the connection with the remote update server
database	PostgreSQL 10.16 database server for 32-bit Linux operating systems. It is required to configure this server and to create and configure the database to be able to store system data and events in a secure way. Size of the database will depend on the number of events that are produced by the system, but initial size is 150 MB
application	Python 3.9.2 compiler for 32-bit Linux operating systems with tkinter (graphical user interface) and psycopg2 (database connection) packages, 100 MB. It is required to develop the application to provide user interface for the operator of the system as well as to communicate with other devices of the system taking the security requirements into account (input data validation for the database and etc.), 100 MB
power bank	RAVPower 20000mAh (near 12 hours of one-board computer work at moderate load), high speed charging, output 5V/3A, 9V/3A, 12V/3A 14.5V/2A, 15V/3A, 20V/3A, 3 hours recharging time

The role of such components is in providing data about the concrete parameters: computing power, disk space, resources consumption, energy consumption, network bandwidth as well as possible incompatibilities between them. These parameters are required for the correct formation of alternatives of the server. Moreover, they are necessary for the ranking of alternatives based on their price, energy efficiency, size and other non-functional characteristics. These calculations will be discussed in more detail in Chapter 3, where the design methodology is presented.

A2. Modeling of charging stations

Each charging station is based on a combination of Iskra JS and ESP8266 microcontrollers with Qi wireless charger. Iskra JS is used to interact with electronic components — sensors and transmitters. This interaction is based on the firmware of the microcontroller and an additional hardware element — Troyka Shield. Troyka Shield does not reserve any pins of the microcontroller. Instead, it provides voltage and ground to each analog and digital pin of the microcontroller simplifying the connection of electronic components. ESP8266 is connected to TxRx pins of Iskra JS and enhances charging station functionality with possibility of remote firmware update on both microcontrollers as well as possibility to connect it to Wi-Fi access points. To provide a power supply, Iskra JS and Qi wireless charger are connected to the home power supply. Iskra JS and Troyka Shield are connected in such a way that they have a shared power supply. Qi transmitter is used to provide a possibility of wireless rechargement of mobile robots.

Thus, in terms of the developed model, bb_{2i} contains the following sub-blocks:

- $bb_1^{'}$ — ESP8266 microcontroller with default drivers and libraries for the possibility to communicate with the server via Wi-Fi;
- $bb_2^{'}$ — Iskra JS microcontroller with default drivers and libraries for the possibility to interact with electronic components via firmware, as well as with $bb_1^{'}$ via AT-commands.

And the following hardware elements:

- hw_1 — Troyka Shield to simplify the connection of other hardware elements and building blocks to $bb_2^{'}$;
- hw_2, hw_3 — two infrared transmitters for the communication with mobile robots during their parking;
- hw_4 — noise sensor for the detection of intruders;
- hw_5 — motion sensor for the detection of intruders.

As well as the following software elements:

- sw_1 — firmware of $bb_1^{'}$ that represents a combination of different algorithms to ensure the possibility of communication between$bb_1^{'}$ and $bb_2^{'}$ as well as bb_{2i} and the server of the system;
- sw_2 — firmware of $bb_2^{'}$ that represents a combination of different algorithms to ensure microcontroller's interaction with transmitters $hw_2 - hw_3$ and sensors

$hw_4 - hw_5$ as well as combination of their output for successful intruder detection and self-availability monitoring; moreover, sw_2 provides $bb_2^{'}$ a possibility to communicate with mobile robots during their parking via $hw_2 - hw_3$ and with $bb_1^{'}$ via AT-commands.

Thus, links between elements of bb_{3i} can be represented as follows:

$$L_{bb_{2i}} = (L_1, L_2, L_3),$$

where L_1 — pin-to-pin connection between $bb_2^{'}$ and $hw_1$with shared power supply; L_2 — TxRx connection between $bb_2^{'}$ and $bb_1^{'}$ with shared ground and voltage wires (four digital pins of Iskra JS are reserved for communication between controllers); L_3— three wire SVG (signal, voltage, ground) connection through hw_1 between $bb_2^{'}$ and transmitters $hw_2 - hw_3$ as well as digital sensors $hw_4 - hw_5$, where digital pins of Iskra JS are used.

FR represents the functionality that satisfaction is necessary for bb_{3i} to be able to work. It contains:

- fr_1 — power source;
- fr_2 — secure communication with the server of the system;
- fr_3 — mobile robots with IR receivers.

NL represents the limitations which satisfaction is necessary for bb_{3i} to be able to work. It contains:

- nl_1 — space for the placement of the device;
- nl_2 — the environment does not contain elements that can lead to incorrect operation of sensors of the device;
- nl_3 — the environment does not contain elements that can lead to incorrect operation of wireless communications.

PFrepresents the functionality that bb_{3i}can provide. It contains:

- pf_1 — to detect intruders;
- pf_2 — to connect to Wi-Fi access points;
- pf_3— to monitor own availability and location;
- pf_4 — to recharge wirelessly mobile robots;

- pf_5 — to receive and install firmware updates remotely;
- pf_6 — to be secure against attackers with $ac = 4$, $kn = 2$, $rs = 2$.

PR represents the resources that bb_{3i} can provide. It contains:

- pr_1 — to add additional software elements to microcontrollers firmware;
- pr_2 — to add, remove or replace hardware elements of charging stations.

Based on the requirements, the algorithm can form the device that represents one of the charging stations step-by-step. For such a device it is required to have:

- microcontroller with a bootloader and programmable firmware that can work with sensors, servo drives, transmitters and shields as well as other microcontrollers, has output voltage 3.3 V on pins and not less than 256 KB of flash memory;
- troyka shield to provide voltage and ground to each analog and digital pin of the microcontroller simplifying the connection of electronic components that are compatible with the selected microcontroller;
- two infrared transmitters for the communication with infrared receivers of mobile robots during their parking process;
- motion sensor with adjustable sensitivity, viewing angle not less than 110 degrees and observing distance not less than 7 m;
- noise sensor with adjustable microphone sensitivity to detect abnormally loud sounds for the detection of intruders;
- servo drive to rotate noise and motion sensors for the better detection of intruders with rotation range not less than 180 degrees;
- microcontroller with a bootloader and programmable firmware with in-build physical Wi-Fi interface and possibility to connect to wireless access points that can have wired data connection with the selected microcontroller;
- Qi transmitter that can be connected to the home power grid;
- firmware of the first microcontroller that contains algorithms for work with all connected electronic components and microcontrollers as well as algorithms that determine the work process of the charging station;
- firmware of the second microcontroller that determines his work with another microcontroller as well as his communication with the server of the system.

But this is without taking into account security requirements. To prevent rpt, rmt and pwr on the level of each charging station a vandal-proof device case (hardware element) should be used. To prevent imw, in addition to the vandal-proof device case, light-weight encryption and authentication algorithms (software elements) for the communication between microcontrollers should be used. To prevent iec, all necessary data for the behavior-based anomaly detection algorithms should be transferred to the designed server of the system. That will provide a possibility to isolate the mobile robot that is under attack. To prevent iws each mobile robot should

be connected to the Wi-Fi access point that is created by the server of the system. To prevent web and dbd on the level of charging stations no additional security elements are required because there are no batteries, web-services or databases.

Thus, the step-by-step formation of each of the charging stations:

1. Combination of the *microcontroller* and *troyka shield* based on the pin-to-pin connection (they should be compatible). At this point the algorithm formed a microcontroller-based platform with a shared power supply to which other components of the charging station can be easily connected.
2. Combination of the *microcontroller with troyka shield* and *microcontroller with in-build Wi-Fi interface* based on the Serial connection (voltage, ground, Tx, Rx). Such a connection will reserve 4 digital pins for the communication between microcontrollers.
3. Combination of the *connected microcontrollers* and *two infrared transmitters* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 2 digital pins (one for each transmitter).
4. Combination of the *microcontrollers with transmitter* and *motion sensor* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
5. Combination of the *microcontrollers with motion sensor* and *noise sensor* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
6. Combination of the *microcontrollers with noise sensor* and *servo drive* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
7. Combination of the *microcontrollers with the motion sensor* and *Qi transmitter* through shared power supply from the home power grid. At this point the algorithm formed a prototype of the device that represents charging stations of the system.
8. Combination of the *charging station prototype* and *firmware of the first microcontroller* based on its programming interface.
9. Combination of the *charging station prototype* and *firmware of the second microcontroller* based on its programming interface.
10. Combination of the *charging station prototype* and *vandal-proof device case* to provide an additional security from physical attacks.

As was mentioned before, firmware of the first microcontroller represents a combination of algorithms for work with all connected electronic components and microcontrollers as well as algorithms that determine the work process of the charging station. In addition, according to security requirements such a firmware must be extended with light-weight encryption and authentication algorithms.

Firmware of the second microcontroller represents a combination of algorithms for work with the first microcontroller as well as for communication with the server of the system. Work with the first microcontrollers should be through the selected

light-weight encryption and authentication algorithms, while the communication with the server must be through its Wi-Fi access point.

The idea behind using two infrared transmitters for directing mobile robots to the charging space is pretty simple. Each charging station with help of these transmitters emits two infrared signals in a V shape. The infrared receiver on each of the mobile robots can receive those signals. Each side of the V is provided by the different transmitter, that is why the mobile robot is able to distinguish them. Thus, when the mobile robot can receive signal only from one of the transmitters it can be interpreted as the “left” or the “right” side of the charging station, so movement can be adjusted. And once signals from both transmitters can be received – the direction to the charging station is known.

One of possible alternatives of the charging station is presented in Table A.6.

Table A.6. Component composition of the charging station

Abstract component	Selected physical device
microcontroller for work with electronic components	Iskra JS with 3.3 and 5V output pins, 1024 KB flash memory, 192 KB SRAM, 14 digital and 6 analog pins and Tx Rx support on P0-P1, 69×53×19 mm size, 300 mAh energy consumption
troyka shield	Troyka Shield with 69×53×19 size that is compatible with Iskra JS and provides 3.3 or 5V on pins
wireless transmitters	infrared transmitter (troyka module), 38 kHz for compatibility with receivers of mobile robots, 25.4×25.4 mm, 20 mAh, 1 digital pin
motion sensor	infrared motion sensor (troyka module), 25.4×25.4 mm, 7 m detection distance, 110 degrees viewing angle, 10 mAh, 1 digital pin
noise sensor	noise sensor (troyka module), 25.4×25.4 mm, 10 mAh, 1 digital pin
servo drive	Feetech FS90 micro servo drive, 180 degrees rotation range, torque 1.3 kg×cm, 650 degrees×sec rotation speed, 150 mAh, 1 digital pin
microcontroller with in-build Wi-Fi interface	ESP8266 (troyka module), UART connection to Iskra JS, 512 KB flash memory, Wi-Fi b/g/n 2.4 GHz, 250 mAh, 4 digital pins
Qi transmitter	5V 0.6A 3W Qi Wireless Charging Coil, micro-USB
firmware for the first microcontroller	It is required to develop the firmware to ensure microcontroller’s work with electronic components as well as combination of their output for successful interaction with mobile robots, intruders and the server. Moreover, light-weight encryption and authentication algorithms are required for communication, 120 KB
firmware for the second microcontroller	It is required to develop the firmware to ensure microcontrollers joint work as well as communication with the server. For communications between microcontrollers light-weight encryption and authentication algorithms are required. Connection with the server must be through its Wi-Fi access point (WPA2-PSK), 100 KB

A3. Modeling of mobile robots

Each mobile robot is based on a combination of Iskra JS and ESP8266 microcontrollers. Iskra JS is used to interact with electronic components — sensors, receivers, servos and motors. This interaction is based on the firmware of the microcontroller and additional hardware elements: Motor Shield and Troyka Shield. Motor Shield provides a possibility to increase the output voltage of the microcontroller with the help of H-bridges which is necessary for correct control of the speed of connected motors. Note that only two motors can be connected to Motor Shield, while 4 digital pins of Iskra JS are reserved for motor control (two pins for each motor to be able to change its movement direction between clockwise and counterclockwise). Troyka Shield does not reserve any pins of the microcontroller. Instead, it provides voltage and ground to each analog and digital pin of the microcontroller simplifying the connection of electronic components. ESP8266 is connected to Tx Rx pins of Iskra JS and enhances mobile robot functionality with possibility of remote firmware update on both microcontrollers as well as possibility to connect it to Wi-Fi access points. To provide a power supply, Motor Shield is connected to a power bank which in turn is connected to Qi wireless charge receiver. Note that Iskra JS, ESP8266, Motor Shield and Troyka Shield are connected in such a way that they have a shared power supply from the power bank. Qi receiver is used to provide a possibility of wireless rechargement of mobile robots.

Thus, in terms of the developed model, bb_{2i} contains the following sub-blocks:

- $bb_1^{'}$ — ESP8266 microcontroller with default drivers and libraries for the possibility to communicate with the server via Wi-Fi;
- $bb_2^{'}$ — Iskra JS microcontroller with default drivers and libraries for the possibility to interact with electronic components via firmware, as well as with $bb_1^{'}$ via AT-commands.

And the following hardware elements:

- hw_1 — Troyka Shield to simplify the connection of other hardware elements and building blocks to $bb_2^{'}$;
- hw_2 — combination of Motor Shield (hardware element) and two collector motors (hardware elements) for the possibility to increase the output voltage of $bb_2^{'}$ on them, so designed mobile robot can move;
- hw_3 — combination of the power bank (hardware element) and Qi receiver (hardware element) that are connected with each other via micro-USB, while hw_3 is connected to hw_2 via two wires (ground and voltage).

- hw_4 — infrared receiver for the communication with charging stations for better parking;
- hw_5 — touch sensor for the detection of obstacles;
- hw_6 — noise sensor for the detection of intruders;
- hw_7 — motion sensor for the detection of intruders;
- hw_8 — distance sensor for the detection of obstacles;
- hw_9, hw_{10} — encoders for wheels that are connected to each of the collector motors of $bb_3^{'}$ to measure the distance traveled by bb_{2i};
- hw_{11} — servo drive for rotation of $hw_6 - hw_8$ sensors for better detection of obstacles and intruders.

As well as the following software elements:

- sw_1 — firmware of $bb_1^{'}$ that represents a combination of different algorithms to ensure the possibility of communication between$bb_1^{'}$ and $bb_2^{'}$ as well as bb_{2i} and the server of the system;
- sw_2 — firmware of $bb_2^{'}$ that represents a combination of different algorithms to ensure its interaction with collector motors hw_2, receiver hw_4, sensors $hw_5 - hw_{10}$and servo drive hw_{11} as well as combination of their output for successful in-door navigation and intruder chase and detection; moreover, sw_2 provides $bb_2^{'}$ a possibility to monitor charge of hw_3, to communicate with charging stations for parking via hw_4 and to communicate with $bb_1^{'}$ via AT-commands.

Thus, links between elements of bb_{2i} can be represented as follows:

$$L_{bb_{2i}} = (L_1, L_2, L_3, L_4, L_5, L_6),$$

where L_1 — pin-to-pin connection between $bb_2^{'}$ and $hw_1$with shared power supply; L_2 — pin-to-pin connection between $bb_2^{'}$ and hw_2 with shared power supply (four digital pins of Iskra JS are reserved for control of two collector motors); L_3 — two-wire VG (voltage, ground) connection between hw_2 and hw_3; L_4 — TxRx connection between $bb_1^{'}$ and $bb_2^{'}$ with shared ground and voltage wires (four digital pins of Iskra JS are reserved for communication between controllers); L_5 — three wire SVG (signal,

voltage, ground) connection through hw_1between $bb_2^{'}$ and receiver hw_4, digital sensors $hw_5 - hw_8$, servo drive hw_{11}, where digital pins of Iskra JS are used; L_3 — three wire SVG connection through hw_1between $bb_2^{'}$ and encoders $hw_9 - hw_{10}$, where analog pins of Iskra JS are used.

Properties of bb_{2i} — one of the mobile robots of the system — are also connected with the properties of mbs and partially represent them.

FR represents the functionality that satisfaction is necessary for bb_{2i} to be able to work. It contains:

- fr_1– secure communication with the server of the system;
- fr_2– availability of wireless charging stations with IR transmitters.

NL represents the limitations which satisfaction is necessary for bb_{2i} to be able to work. It contains:

- nl_1 — space for mobile robots' movement;
- nl_2 — environment does not contain elements that can lead to incorrect operation of device sensors;
- nl_3 — environment does not contain elements that can lead to incorrect operation of wireless communications.

PF represents the functionality that bb_{2i}can provide. It contains:

- pf_1– to detect and avoid obstacles;
- pf_2– to detect and chase intruders;
- pf_3– to connect to Wi-Fi access points;
- pf_4– to monitor own position inside controlled perimeter;
- pf_5– to monitor the charge state of the power bank;
- pf_6– to recharge wirelessly on charging stations;
- pf_7– to receive and install firmware updates remotely;
- pf_8– to be secure against attackers with $ac = 4, kn = 2, rs = 2$.

PR is representing the resources that bb_{2i} can provide. It contains:

- pr_1– to add additional software elements to microcontrollers firmware;
- pr_2– to add, remove or replace hardware elements of mobile robots.

As an example of the hardware element that contains hardware sub-elements, let's consider $hw_2 \in bb_{2i}$ — combination of Motor Shield and two collector motors of one of the mobile robots — in more detail:

$$hw_2 = ((hw_{21}, hw_{22}, hw_{23}),\ L_{hw_2},\ p_{hw_2}),$$

where hw_{21} — Motor Shield of hw_2; hw_{22} — first collector motor of hw_2; hw_{23} — second collector motor of hw_2; L_{hw_2} — VG connections between hw_{21} and $hw_{22} - hw_{23}$; p_{hw_2} — properties of hw_2.

Let's consider properties of $hw_2 \in bb_{2i}$ in more detail. FR represents the functionality that satisfaction is necessary for hw_2 to be able to work. It contains:

- fr_1 — power source;
- fr_2 — physical connection to the microcontroller with correct voltage.

NL is representing the limitations which satisfaction is necessary for hw_2:

- nl_1 — microcontroller shape must be suitable for Motor Shield installation;
- nl_2 — motors must be DC with a voltage of 5 to 24 V;
- nl_3 — input voltage of the power supply in the range from 7 to 12 V;
- nl_4 — power supply provides a stable voltage during sudden load surges.

PF is representing the functionality that bb_{3i} can provide. It contains:

- pf_1 — to control two collector motors (direction and speed of rotation).

PR is representing the resources that hw_2 can provide. It contains:

- pr_1 — to add, remove or replace hardware elements.

Note that properties of $hw_2 \in bb_{2i}$ — combination of Motor Shield and two collector motors — are connected with the properties of $bb_{2i} \in BB_2$ — one of the mobile robots of the designed system — and partially represent them. While properties of $bb_{2i} \in BB_2$ are connected with the properties of the designed mbs.

As an example of the software element that contains software sub-elements, let's consider $sw_2 \in bb_{2i}$ — firmware of Iskra JS microcontroller of one of the mobile robots — in more detail:

$$sw_2 = ((sw_{21},\ldots, sw_{31}), L_{sw_2}, p_{sw_2}),$$

where sw_{21} — software element for control of collector motors via Motor Shield (speed, rotation direction); sw_{22} — software element for processing of data from infrared receivers of charging stations; sw_{23} — software element for processing of data from noise sensor; sw_{24} — software element for processing of data from motion sensor; sw_{25} — software element for processing of data from motion sensor; sw_{26} — software element for processing of data from distance sensor; sw_{27} — software element for control of the servo drive; sw_{28} — software element for processing of data from encoders; sw_{29} — software element for communication with EPS8266 microcontroller; sw_{30} — software element for monitoring of charge state of a power bank; sw_{31} — a software element that works with other elements based on rules of the device behavior (for example, in-door navigation, detection and chase of an attacker, data transferring to the server); L_{sw_2} — links between software elements of sw_2 based on compilation of the source code; p_{sw_2} — properties of sw_2.

Let us consider properties of $sw_2 \in bb_{2i}$ in more detail. FR represents the functionality that satisfaction is necessary for sw_2 to be able to work. It contains:

- fr_1 – microcontroller with bootloader;
- fr_2 – microcontroller with flash memory.

NL represents the limitations which satisfaction is necessary for sw_2:

- nl_1 — flash memory space is not less than 256 KB;
- nl_2 — RAM size is not less than 64 KB.

PF represents the functionality that sw_2 can provide. It contains:

- pf_1 — in-door navigation of mobile robots;
- pf_2 — monitoring of charge state of the mobile robot;
- pf_3 — detection of charging stations and parking;
- pf_4 — detection and chase of intruders;
- pf_5 — communication with the server of the system.

PR represents the resources that sw_2 can provide. It contains:

- pr_1— to add, remove or replace software elements;
- pr_2— to change mobile robots behavior;
- pr_3— to update microcontrollers firmware.

Note that properties of $sw_2 \in bb_{2i}$ — firmware of Iskra JS microcontroller of one of the mobile robots — are connected with properties of $bb_{2i} \in BB_2$ — one of the mobile robots of the designed system — and partially represent them. While properties of $bb_{2i} \in BB_2$ are connected with the properties of the designed mbs.

Based on the requirements, the algorithm can form the device that represents one of the mobile robots step-by-step. For such a device it is required to have:

- microcontroller with a bootloader and programmable firmware that can work with sensors, collector motors, servo drives, transmitters, receivers and shields as well as other microcontrollers, has output voltage 3.3 V on pins and not less than 256 KB of flash memory;
- two collector motors to rotate wheels of the robot with not less than 300 rpm and size equal to 12 mm;
- motor shield to increase the output voltage of the selected microcontroller with help of H-bridges that is able to control not less than two motors and compatible with the selected microcontroller;
- troyka shield to provide voltage and ground to each analog and digital pin of the microcontroller simplifying the connection of electronic components that are compatible with the selected microcontroller;
- wireless receiver for the communication with two directed wireless transmitters of charging stations during parking that is compatible with the selected microcontroller (the choice here determines the choice for each charging station);
- touch sensor for the detection of obstacles that can be used as bumper and compatible with the selected microcontroller;
- distance sensor for the detection of obstacles with distance range not less than 400 cm and effective viewing angle not less than 15 degrees that is compatible with the selected microcontroller;
- noise sensor with adjustable microphone sensitivity to detect abnormally loud sounds for the detection of intruders;
- motion sensor with adjustable sensitivity, viewing angle not less than 110 degrees and observing distance not less than 7 m;
- servo drive to rotate distance, noise and motion sensors for the better detection of obstacles and intruders due to viewing angle limits with rotation range not less than 180 degrees;

- encoders for wheels that are connected to each of the selected collector motors to measure the distance traveled by the designed robot;
- microcontroller with a bootloader and programmable firmware with in-build physical Wi-Fi interface and possibility to connect to wireless access points that can have wired data connection with the selected microcontroller;
- power bank that can power up the combination of selected microcontrollers, shields, sensors, motors, servos and receivers and has a capacity to provide power supply to the designed robot for not less than 2 hours;
- Qi receiver that can be connected to the selected power bank (the choice here determines the choice for each charging station);
- firmware of the first microcontroller that contains algorithms for work with all connected electronic components and microcontrollers as well as algorithms that determine the work process of the mobile robot;
- firmware of the second microcontroller that determines his work with another microcontroller as well as his communication with the server of the system.

But this is without taking into account security requirements.

To prevent rpt and rmt on the level of each mobile robot a vandal-proof device case (hardware element) should be used.

To prevent imw, in addition to the vandal-proof device case, light-weight encryption and authentication algorithms (software elements) for the communication between microcontrollers should be used.

To prevent iec, all necessary data for the behavior-based anomaly detection algorithms should be transferred to the designed server of the system. That will provide a possibility to isolate the mobile robot that is under attack.

To prevent iws each mobile robot should be connected to the Wi-Fi access point that is created by the server of the system.

To prevent pwr, web and dbd on the level of each mobile robot no additional security elements are required because there are no web-services or databases.

Thus, the step-by-step formation of each of the mobile robots:

1. Combination of *motor shield* and *two collector motors* with the help of the two-wire connection (voltage and ground).
2. Combination of the *power bank* and *Qi wireless charge receiver* based on the micro-USB connection.
3. Combination of the *motor shield with motors* and *power bank with the receiver* based on two-wire connection (voltage and ground).

4. Combination of the *motor shield with power bank* and *microcontroller* based on the pin-to-pin connection (selected shield and microcontroller should be compatible). Such a connection will reserve 4 digital pins.
5. Combination of the *microcontroller with motor shield* and *troyka shield* based on the pin-to-pin connection (selected shield and microcontroller should be compatible). At this point the algorithm formed a microcontroller-based platform with a shared power supply to which other components of the mobile robot can be easily connected.
6. Combination of the *microcontroller with troyka shield* and *microcontroller with in-build Wi-Fi interface* based on the Serial connection (voltage, ground, Tx, Rx). Such a connection will reserve 4 digital pins for the communication between microcontrollers.
7. Combination of the *connected microcontrollers* and *wireless receiver* based on the three-wire connection (voltage, ground, signal). Selected at this point, the receiver will determine the choice of transmitters for charging stations. Such a connection will reserve 1 digital pin.
8. Combination of the *microcontrollers with the receiver* and *touch sensor* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
9. Combination of the *microcontrollers with the touch sensor* and *distance sensor* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
10. Combination of the *microcontrollers with the distance sensor* and *noise sensor* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
11. Combination of the *microcontrollers with the noise sensor* and *motion sensor* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
12. Combination of the *microcontrollers with motion sensor* and *servo drive* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 1 digital pin.
13. Combination of the *microcontrollers with servo drive* and *two encoders* based on the three-wire connection (voltage, ground, signal). Such a connection will reserve 2 analog pins. At this point the algorithm formed a prototype of the device that represents mobile robots of the system.
14. Combination of the *mobile robot prototype* and *firmware of the first microcontroller* based on its programming interface.
15. Combination of the *mobile robot prototype* and *firmware of the second microcontroller* based on its programming interface.
16. Combination of the *mobile robot prototype* and *vandal-proof device case* to provide an additional security from physical attacks.

One of possible alternatives of the mobile robot is presented in Table A.7.

Table A.7. Component composition of the mobile robot

Abstract component	Selected physical component
microcontroller for work with electronic components	Iskra JS with 3.3 and 5V output pins, 1024 KB flash memory, 192 KB SRAM, 14 digital and 6 analog pins and Tx Rx support on P0-P1, 69×53×19 mm size, 300 mAh energy consumption
collector motors	12mm motor, 5V, 300 rpm, 3 mm shaft diameter, 10 mm shaft length, 36×12×10 mm size, 50 mAh energy consumption
motor shield	two channeled Motor Shield based on L298P that supports connection of two collector motors, 2A, 5-12V, 4 digital pins
troyka shield	Troyka Shield with 69×53×19 size that is compatible with Iskra JS and provides 3.3 or 5V on pins
wireless receiver	infrared receiver in form of troyka module (can be easily connected to Troyka Shield), 38 kHz, 25.4×25.4 mm, 20 mAh, 1 digital pin
touch sensor	clock button in form of troyka module (can be easily connected to Troyka Shield), 25.4×25.4 mm, 5 mAh, 1 digital pin
distance sensor	ultrasonic distance sensor HC-SR04, 2-400 cm scanning distance, 15 degrees effective viewing angle, 15 mAh, 1 digital pin
noise sensor	noise sensor in form of troyka module (can be easily connected to Troyka Shield), 25.4×25.4 mm, 10 mAh, 1 digital pin
motion sensor	infrared motion sensor in form of troyka module (can be easily connected to Troyka Shield), 25.4×25.4 mm, 7 m detection distance, 110 degrees viewing angle, 10 mAh, 1 digital pin
servo drive	Feetech FS90 micro servo drive, 180 degrees rotation range, torque 1.3 kg×cm, 650 degrees×sec rotation speed, 150 mAh, 1 digital pin
encoders for wheels	line sensor based on TCRT5000, 10 mAh, 1 analog pin
microcontroller with in-build Wi-Fi interface	ESP8266 in from of troyka module (can be easily connected to Troyka Shield), UART connection to Iskra JS, 512 KB flash memory, Wi-Fi b/g/n 2.4 GHz, 250 mAh, 4 digital pins
power bank	Power Bank v2 2000 mAh, 5V, 600 mA, 55×53×20 mm, micro-USB, two-wire (ground + voltage) connection
Qi receiver	5V 0.6A 3W Qi Wireless Charging Coil Receiver, micro-USB
firmware for the first microcontroller	It is required to develop the firmware to ensure microcontroller's work with electronic components as well as combination of their output for successful in-door navigation and interaction with charging stations, the server and intruders. Moreover, light-weight encryption and authentication algorithms are required for communication, 520 KB
firmware for the second microcontroller	It is required to develop the firmware to ensure microcontrollers joint work as well as communication with the server of the system. For communications between microcontrollers light-weight encryption and authentication algorithms are required. Connection with the server must be through its Wi-Fi access point (WPA2-PSK), 330 KB

Appendix B. Verification of mobile robots

This appendix describes our experience in verification of the two-wheel mobile robot that is based on the LEGO 9797 Mindstorms NXT controller. It contains input data description, as well as our experience in work with the SPASS theorem prover, the Maude system and the daTac system.

B1. Description of the input data

Main elements of the LEGO 9797 Mindstorms NXT controller as well as its communication protocols and interfaces can be represented as follows:

- *battery*: six AA/LR6 batteries or one special rechargeable lithium-ion battery;
- *LCD*: graphical user interface and possibility to output visual information;
- *speaker*: possibility to play different sounds;
- *ports A, B, C*: possibility to connect motors;
- *ports 1, 2, 3 or 4*: possibility to connect sensors;
- *USB*: possibility to connect the controller to the computer (wire connection), update its firmware, send scripts and receive commands from the computer;
- *Bluetooth*: possibility to connect the controller to the computer or other device (wireless connection), update its firmware, send scripts and receive commands from the computer or other device.

Motors are connected to the controller via ports A, B and C using an RJ12 cable on a 1 to 1 basis, so the controller can simultaneously work with no more than three motors. These motors are servos for which one rotation is equal to 360 degrees, while there is a possibility to program them to rotate on a certain number of degrees. The operation of each motor is programmed through the port to which it is connected, while it is possible to set operation scenarios for a single one or several. For example, it is possible to rotate a robot with a simple algorithm that works only with its chassis: to rotate to the left, it is required to stop the motor of the left part of the chassis while the motor of the right part should continue to work.

Sensors can be connected to ports 1, 2, 3 and 4 in a similar way, so the controller can simultaneously work with no more than four sensors. Work with sensors is based on the reaction to their events. Let’s consider them in more detail:

- *touch sensor*: works on the principle of a clock button, allows one to track the transition from the pressed state to the original one and vice versa;
- *sound sensor*: works on the principle of a microphone, allows one to measure the noise level nearby the robot;
- *light sensor*: allows one to evaluate the brightness of the object, based on which it is possible to distinguish colors;
- *distance sensor (ultrasonic)*: allows one to measure the distance from the robot to the obstacle in the range from 0 to 255 centimeters.

Interaction with motors during their programming is presented in Figure B.1.

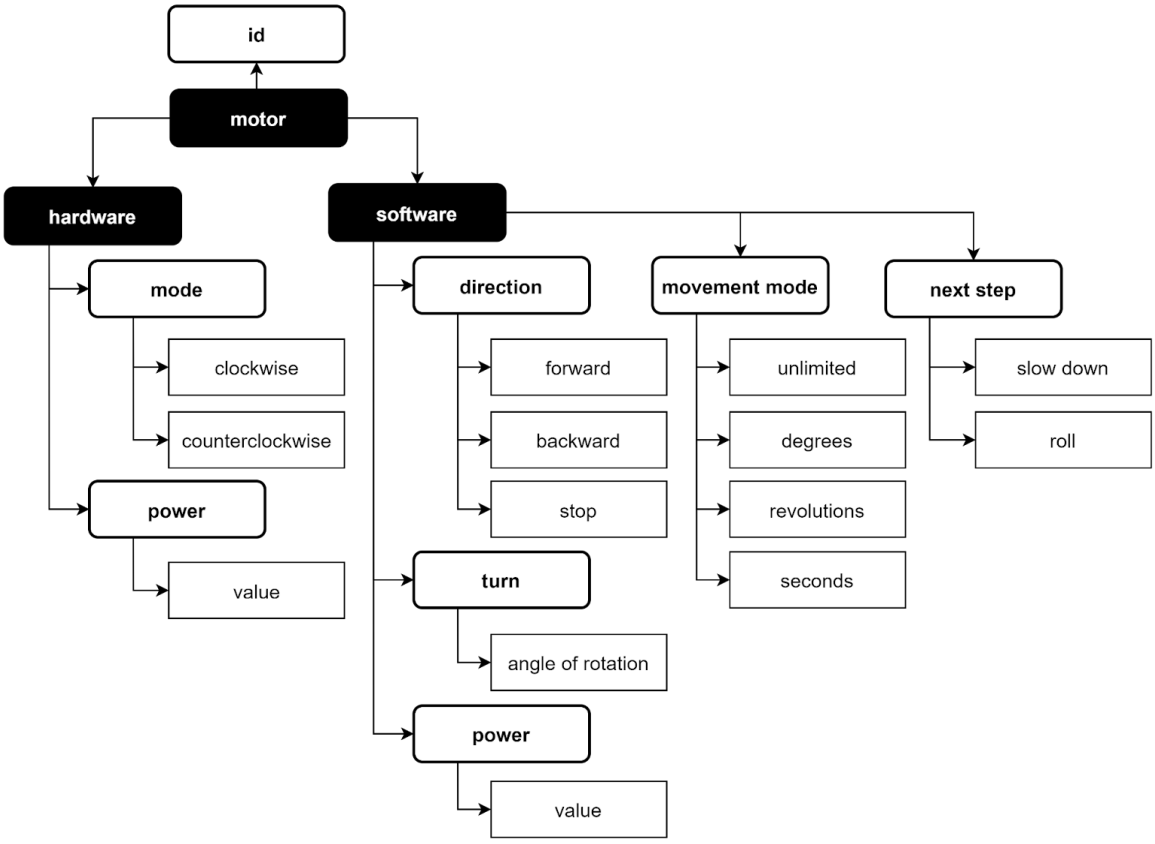

Figure B.1. An overview of the LEGO motor model

Interaction with sensors during their programming is presented in Figure B.2.

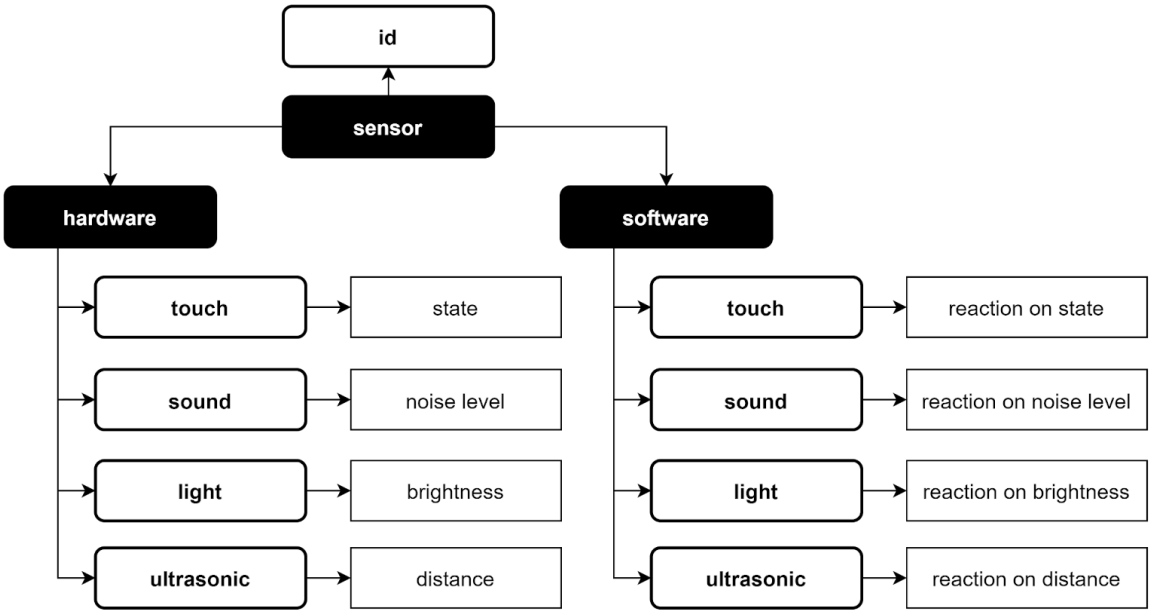

Figure B.2. An overview of the LEGO sensors model

For example, the touch sensor can be used for the development of the obstacle detection element — a bumper. The bumper can be used to prevent the collision of a mobile robot with an obstacle: if touch sensor state is pressed then stop to move. Sound sensor can be used to control a mobile robot using sound signals: if the noise level is more than some threshold then start to move. Light sensor can be used to detect a line and to move along it: if the brightness of the object is more than some threshold then rotates to the left, else — to the right. Moreover, the light sensor can be used to prevent the mobile robot from falling off the table and to count the number of wheel revolutions. Distance sensor can be used to control the distance between the mobile robot and the detected obstacle: if distance is less than some threshold then rotates to the left. In more complicated scenarios, actions of the mobile robot are dependent on the state of several sensors at once: the mobile robot needs to detect an obstacle, come closer to it and stop at a certain distance to check the color of the object near the obstacle and to take it or not depending on its color.

Note that any mobile that is designed on the basis of the LEGO Mindstorms constructor is a combination of a controller, motors, sensors and a large number of LEGO parts that make it possible to build its case with all moving elements. At the same time, the programming of the controller can be carried out both in a special graphical environment and by writing the source code. Moreover, the presence of the wireless interface in the controller allows one to build not only individual robots but also organize their interaction with other robots as well as the human operator.

B2. Description of experiments

For the experiments, it was decided to verify the possibility of designing a mobile robot with only one ability — ability to move. In its turn, the ability to move is granted to robots that have a battery, wheels and a motor. The goal was to check existing tools for verification and try to adapt them for automated design, namely, the SPASS theorem prover, the Maude system and daTac. They were used to investigate different possibilities of reaching a mobile robot based on available building blocks.

For each of these tools appropriate specifications were developed. Their full descriptions are available on https://github.com/levshun/PhD-mcbpss_design. Let's consider each tool and developed model in more detail.

Firstly, the problem was encoded for the SPASS — an automated theorem prover for first-order logic with equality. The result was obtained in clause number 525:

```
Given clause: 525[0:Res:517.0,12.0] ||
 has(battery,plug(plug(plug(chassis0,battery0), motor0),wheel0))
 has(motor,plug(plug(plug(chassis0,battery0),motor0),wheel0))
 has(wheel,plug(plug(plug(chassis0,battery0),motor0),wheel0))* -> .
 SPASS V 3.9
 SPASS beiseite: Proof found.
```

The experiment showed that SPASS does not support associativity-commutativity unification, so it was decided to develop a small hack with `plugable` as a list constructor. But there are side effects:

- deduction process is faster than usual;
- system will diverge anyway by paramodulation into the `plug` formula;
- developed model will be incomplete if the system can have male/female slots.

After SPASS, the Maude system was used. Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification for a wide range of applications. The result was obtained almost immediately:

```
search in CPS :
 init =>* S | system(Id1, P + has(wheel) + has(motor) + has(battery)) .
 Solution 1 (state 0)
 states: 1 rewrites: 4 in 0ms cpu (0ms real) (~ rewrites/second)
 S --> none
 Id1 --> plug(wheel, plug(motor, plug(battery, chassis)))
 P --> done

 No more solutions.
 states: 1 rewrites: 4 in 0ms cpu (0ms real) (~ rewrites/second)
```

The initial state was represented as a multiset of resources, while each resource has one or more interfaces using which it can be attached to another resource. Also, each resource provides some properties, such as having a battery or wheels. The goal is that the set of options contains one that has a conjunction of properties. Since the main focus was on the physical part of the composition, there is only one composition rule for components that consists in attaching to components together. This operation consumes the existing components and creates a new one. The `search` functionality of the Maude system was also used to let it explore all the possible combinations that may lead to the construction of the mobile robot with the correct set of properties.

The aim of the daTac system is to do automated deduction in first-order logic with equality. Its speciality is to apply deductions modulo some equational properties of operators, such as commutativity or associativity-commutativity. The result was obtained pretty fast:

```
daTac -x o -i cps_composition -o cps_composition

 Clause 3: => composition(system(slot(male,wheel,id(wheel,x1)).empty,has(wheel)
 .empty,x1).system(slot(male,motor,id(motor,x2)).empty,has(motor).empty,x2)
 .system(slot(male,battery,id(battery,x3)).empty,has(battery).empty,x3)
 .system(slot(female,wheel,id(chassis,x4)).slot(female,motor,id(chassis,x4))
 .slot(female,battery,id(chassis,x4)).empty,empty,x4).empty,needs(wheel)
 .needs(motor).needs(battery).empty,empty,empty)

 Clause 21: => composition(system(slot(male,wheel,id(wheel,s(x1))).empty,
 has(wheel).empty,x1).empty.system(slot(female,wheel,id(chassis,x2))
```

```
.slot(female,motor,id(chassis,s(x2))).slot(female,battery,id(chassis,s(x2)))
.empty,empty,x2).system(slot(male,motor,id(motor,x3)).empty,has(motor).empty,x3)
.system(slot(male,battery,id(battery,x4)).empty,has(battery).empty,x4),
empty.needs(battery).has(battery).empty,empty.empty,empty.plug(id(wheel,s(x1))
,wheel,id(chassis,s(x2))).plug(id(motor,s(x3)),motor,id(chassis,s(x2)))
.plug(id(battery,s(x4)),battery,id(chassis,s(x2))))

Simplification from 2 into 21

Clause 21: => composition(system(slot(male,wheel,id(wheel,s(x1))).empty,
has(wheel).empty,x1).empty.system(slot(female,wheel,id(chassis,x2))
.slot(female,motor,id(chassis,s(x2))).slot(female,battery,id(chassis,s(x2)))
.empty,empty,x2).system(slot(male,motor,id(motor,x3)).empty,has(motor).empty,x3)
.system(slot(male,battery,id(battery,x4)).empty,has(battery).empty,x4),
empty.needs(battery).has(battery),empty,empty.plug(id(wheel,s(x1)),wheel,
id(chassis,s(x2))).plug(id(motor,s(x3)),motor,id(chassis,s(x2)))
.plug(id(battery,s(x4)),battery,id(chassis,s(x2))))

Simplification from 1 into 21

Clause 21: => composition(system(slot(male,wheel,id(wheel,s(x1))).empty,
has(wheel).empty,x1).empty.system(slot(female,wheel,id(chassis,x2))
.slot(female,motor,id(chassis,s(x2))).slot(female,battery,id(chassis,s(x2)))
.empty,empty,x2).system(slot(male,motor,id(motor,x3)).empty,has(motor).empty,x3)
.system(slot(male,battery,id(battery,x4)).empty,has(battery).empty,x4),
empty.empty,empty,empty.plug(id(wheel,s(x1)),wheel,id(chassis,s(x2)))
.plug(id(motor,s(x3)),motor,id(chassis,s(x2))).plug(id(battery,s(x4)),
battery,id(chassis,s(x2))))

Clausal Simplification in 21 thanks to 8
Simplification from 2 into 21
Clause 21: []

...
Total User Time: 0.005485 s cptTimbnbe: 50
```

The rules encoding the composition problem in daTac are similar to those employed with Maude but for the fact that problems have to be encoded in first-order logic and that the composition strategy tries to simplify a composition problem denoted with the `composition` 4-ary predicate symbol. The first argument is the multiset of different kinds of components. The second one lists slots available for aggregating new components. The third component lists functional requirements on the goal of the design process — a mobile robot. Finally, the last component is employed for bookkeeping to trace which and how components were added.

The motivation for considering daTac in addition to Maude is that in spite of its almost deprecated status, it is the theorem prover that handles natively AC-unification, and, thus, searches all possible combinations of components without an additional encoding. Being a theorem prover, it is not committed to a forward search of all combinations from the initial state (Clause 3), that is why it was possible to implement a lazy search strategy based on an ordered strategy.

Appendix C. Extraction of vulnerabilities of devices

This appendix describes our experience in the connection of descriptions of devices with CPE URIs in accordance with their hardware, software and firmware. In addition, this appendix shows how the obtained list of CPE URIs that represents the configuration of the device can be checked for being vulnerable and connected with CVE descriptions.

C1. Extraction of CPE URIs

CPE URI of version 2.3 contains the following fields: *part*, *vendor*, *product*, *version*, *update*, *edition*, *language*, *sw_edition*, *target_sw*, *target_hw* and *other*. For example:

```
[part="o",vendor="microsoft",product="windows_vista",version="6\.0",
update="sp1",edition=NA,language=NA,sw_edition="home_premium",
target_sw=NA,target_hw="x64",other=NA]

cpe:2.3:o:microsoft:windows_vista:6.0:sp1:-:-:home_premium:-:x64:-
```

It means that based on the description of the device hardware, software and firmware it is possible to extract the corresponding CPE URIs. Let's consider the main issues of such a process in more detail.

Issues of CPE URIs:

- can contain typos;
- can become deprecated;
- can contain information about the vendor in the *product* field;
- open databases of CVE and CPE are not synchronized (corrections in one of the databases might not be implemented in another, some CPEs might not be connected with CVEs and some CVEs might not be connected with CPEs);
- two different CPE URIs might be equal to each other;
- *version*, *update*, *edition* and *sw_edition* fields can be equal to "-", which can be interpreted as any version, update or edition;
- *version*, *update*, *edition*, *language*, *sw_edition*, *target_sw*, *target_hw* and *other* fields can be equal to "*", which can be interpreted as empty field;
- *version* field can contain special symbol "*" to interpret the range of versions, for example, 8.1* contains 8.123.

Issues of device descriptions:

- output format is most likely not equal to CPE URI;
- content of the output is most likely not equal to the content of CPE URI fields;
- can contain typos;
- output format can change.

It means that the mapping of descriptions to CPE URIs requires an approach that:

- generates multiple options of CPE URI fields for each component of the device in accordance with its description;
- checks the probability of each option to be represented as one of CPE URIs taken into account possibility of typos and other issues mentioned;
- selects the most reasonable option among possible.

Note that such an approach is impossible without *false positives* — device was linked to the wrong CPE URI, and *false negatives* — device was not linked to the correct CPE. The output of such an approach contains the list of CPE URIs that are representing the configuration of the analyzed device. This configuration can be checked in terms of being vulnerable. For more information, see the next section.

C2. Extraction of CVE descriptions

In NVD (National Vulnerability Database) CVE descriptions of vulnerabilities are connected with configurations of devices that are vulnerable to them. Each configuration is represented as an expression with OR and AND operators, that are connecting CPE URIs together in hierarchical structures. For example, the vulnerability CVE-2020-11241 has the following configuration:

```
AND(OR({"cpe23Uri": "cpe:2.3:o:qualcomm:ipq8076_firmware:-:*:*:*:*:*:*:*", "cpe_name":
[], "vulnerable": true}),OR({"cpe23Uri": "cpe:2.3:h:qualcomm:ipq8076:-:*:*:*:*:*:*:*",
"cpe_name": [], "vulnerable": false}))
```

It can be interpreted as follows:

- Qualcomm ipq8076 firmware is vulnerable to CVE-2020-11241 if it is installed on Qualcomm ipq8076 hardware.

Note that configurations can be much bigger than that, for example CVE-2020-0554 is connected with the following one:

```
AND(
        OR(
                {"cpe23Uri": "cpe:2.3:o:intel:ac_3165_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_3168_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_7265_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_8260_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_8265_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9260_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9461_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9462_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
```

```
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9560_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ax200_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"},
                {"cpe23Uri": "cpe:2.3:o:intel:ax201_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.40"}
        ),
        OR(
                {"cpe23Uri": "cpe:2.3:o:microsoft:windows_7:-:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": false},
                {"cpe23Uri": "cpe:2.3:o:microsoft:windows_8.1:-:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": false}
        )
)

AND(
        OR(
                {"cpe23Uri": "cpe:2.3:o:intel:ac_3165_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_3168_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_7265_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_8260_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_8265_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9260_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9461_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9462_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ac_9560_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ax200_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"},
                {"cpe23Uri": "cpe:2.3:o:intel:ax201_firmware:*:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"}
        ),
        OR(
                {"cpe23Uri": "cpe:2.3:o:microsoft:windows_10:-:*:*:*:*:*:*:*",
                "cpe_name": [], "vulnerable": false}
        )
)
```

The process of such configurations checking is based on the replacement of the following data structures:

```
        {"cpe23Uri": "cpe:2.3:o:intel:ax200_firmware:*:*:*:*:*:*:*:*",
        "cpe_name": [], "vulnerable": true, "versionEndExcluding": "21.70"}
```

with “1” and “0” based on the presence of CPE URIs in the configuration of the device. After that each “OR” or “AND” expression is checked individually starting from inner ones. Expressions are also replaced with “0” and “1”, until there is nothing to replace. And if the result is “1” — device is vulnerable, “0” — not vulnerable.

An example of the Python implementation of such a functionality:

```
def check_configuration(uris, config):
 config = config \
 .replace('true', '"True"') \
 .replace('false', '"False"')

 value = ''

 while '(' in config:
 while '{' in config:
 result = re.search(r"\{(.*?)\}", config)
 sub_str = config[result.start():result.end()]

 cpe_dict = \
 ast.literal_eval(sub_str)

 if cpe_dict['cpe23Uri'] in uris:
 config = config.replace(sub_str, '1')
 else:
 config = config.replace(sub_str, '0')

 result = re.search(r'\(([^()]+)\)', config)
 expression = config[result.start()-2:result.end()]
 if expression[:2] == 'ND':
 expression = 'A' + expression

 value = check_expression(expression)
 config = config.replace(expression, value)

 value = bool(int(value))

 return value
```

This implementation checks expressions with the help of the following function:

```
def check_expression(expression):

 operator = expression.split('(')[0]

 values_str = expression \
 .replace(operator, '') \
 .replace('(', '').replace(')', '')

 values_arr = values_str.split(',')

 value = bool(int(values_arr[0]))

 if len(values_arr) > 1:
 for index in range(1, len(values_arr)):
 if operator == 'OR':
 value = value or bool(int(values_arr[index]))
 if operator == 'AND':
 value = value and bool(int(values_arr[index]))

 value = str(int(value))

 return value
```