KAUNAS UNIVERSITY OF TECHNOLOGY

NERIJUS JUSAS

Feature Model-Based Development of Internet of Things Applications

Doctoral dissertation
Technological sciences, Informatics Engineering (07T)

2017, Kaunas

This doctoral dissertation was prepared at Kaunas University of technology, Faculty of Informatics, Department of Computer Science, during the period of 2012 – 2017.

Scientific supervisor:

Prof. dr. Algimantas Venčkauskas (Kaunas University of Technology, technology sciences, informatics engineering – 07T).

Doctoral dissertation has been published in:
http://ktu.edu

Editor:
Dr. Armandas Rumšas (Publishing Office "Technologija")

ISBN 978-609-02-1341-4

The bibliographic information about the publication is available in the National Bibliographic Data Bank (NBDB) of the Martynas Mažvydas National Library of Lithuania

KAUNO TECHNOLOGIJOS UNIVERSITETAS

NERIJUS JUSAS

Požymių modeliais grindžiamas daiktų interneto sistemų kūrimas

Daktaro disertacija
Technologijos mokslai, Informatikos inžinerija (07T)

2017, Kaunas

Disertacija rengta 2012-2017 metais Kauno technologijos universiteto informatikos fakultete kompiuterių katedroje.

Mokslinis vadovas:

Prof. dr. Algimantas Venčkauskas (Kauno technologijos universitetas, technologijos mokslai, informatikos inžinerija – (07T).

Interneto svetainės, kurioje skelbiama disertacija, adresas:
http://ktu.edu

Redagavo: Dr. Armandas Rumšas (leidykla "Technologija")

ISBN 978-609-02-1341-4

Leidinio bibliografinė informacija pateikiama Lietuvos nacionalinės Martyno Mažvydo bibliotekos Nacionalinės bibliografijos duomenų banke (NBDB)

Contents

List of Figures

List of Tables

GLOSSARY

Advanced encryption standard (AES)	Asymmetric block cipher used to protect classified information.
Body Area Network (BAN)	Wireless network of wearable computing devices.
Confidential (C)	Information that could cause risk of material harm to individuals or if disclosed.
Domain space exploration (DSE)	Process to obtain an optimally performing architecture with respect to some constrains.
Domain-specific language (DSL)	Computer language specialized to a particular application domain.
Feature model (FM)	Compact representation of the products of the product line in terms of some "features".
Feature-oriented domain analysis (FODA)	Domain analysis method which introduced feature modelling to domain engineering.
Heterogeneous concentrator (HC)	A device which provides communication capability between many low-speed, usually asynchronous channels and one or more high-speed, usually synchronous channels
Internet of Things (IoT)	The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.
IoT application data collection module (IoTADCM)	Module of IoT application, which is responsible for data processing.
Internet Protocol address (IP)	Numerical label assigned to each device participating in a computer network that uses the Internet protocol for communication.
Measurement-Communication Module (MCM)	Module of IoT application which is responsible for data collection and transmission to a heterogeneous concentrator.
Model-driven engineering (MDE)	An application development methodology that focuses on creating and exploiting domain models, which are conceptual models of all the topics related to a specific problem.
Model-driven development(MDD)	A paradigm for writing and implementing computer programs quickly, effectively and at minimum cost.
Model-driven application development (MDAD)	A concept of being able to make any kind of change to a model as well as to the code generated from that model.

Model-driven product line (MDPL)	A combined paradigm of MDE and SPLE.
Message integrity check (MIC)	A security improvement for encryption found on wireless networks.
Product line (PL) or Product Line Engineering (PLE)	Application engineering methods, tools and techniques for creating a collection of similar applications from a shared set of applications assets by using a common means of production.
Quality of Service (QoS)	Network's ability to achieve maximum bandwidth and deal with other network performance elements like latency, error rate and uptime.
Restricted (R)	Restricted information is the kind of information, the disclosure of which would not cause material harm, but which has been chosen to be kept confidential.
Secret (S)	Information that would cause severe harm to some individuals/entities if disclosed.
Standard Internet Module (SIM)	Module of IoT application, which is responsible for data transmission between MCM and IoTADCM.
Sensitive but Unclassified (SU)	Data that is not considered vital to individuals security, but its disclosure would do some harm.
Simple XML Feature Model format (SXFM)	A mark-up language that is used by an S.P.L.O.T. tool to present feature models.
Top Secret (TS)	Information that would cause severe harm to some individuals/institutions if disclosed.
Unclassified (U)	Data that has no classification or is not sensitive.
Unified Modelling Language (UML)	A standardized modelling language enabling developers to specify, visualize, construct and document artefacts of a software system.
Wireless Personal Area Network (WPAN)	A network for interconnecting devices centered around an individual person's workspace in which the connections are wireless.
Wireless Sensor Network (WSN)	Spatially distributed autonomous sensors designed monitor physical or environmental conditions.
Extensible Mark-up Language (XML)	A mark-up language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

1. INTRODUCTION

1.1. Relevance of the work

Technology advances have resulted in our modern-day life and work in a digital world surrounded with the modern technology infrastructure – multiple devices integrated within networks along with computers, mobile devices, sensor networks, etc. which have become a commodity of our contemporary lives. In the nearest future, however, not only humans and computers but also everyday life items will be interconnected to create the new computing infrastructure – The Internet of Things (IoT) [Atzori et al., 2010]. The goal of the IoT is to enable things to be connected "any-time, any-place, with anything and anyone ideally using any path/network and any service" [Smith, 2012]. Semantically, the IoT means a new highly heterogeneous worldwide network of interconnected objects uniquely addressable, based on standard communication protocols [Labiod et al., 2007]. This move from 'interconnected computers' to 'interconnected things' is a great challenge for the Information-Communication Technology (ICT) practitioners, scientists and the society in the whole. As a response to the challenge, an extremely wide stream of research is being provided worldwide.

In the case of its application, the IoT is to be considered as an information-communication technology with smart features and enhanced capabilities. From this viewpoint, the IoT stands for a huge infrastructure containing physical objects such as sensors, actuators, etc. that are self-identifiable to other devices; thus, being connected over the Internet, these objects enable communication and continuous transmission of the collected or control data over the nodes of a network via the Internet. Depending on the application, diverse sensors and other devices are served in collecting the data, on whose basis, the functionality of the application is built.

Typically, security/privacy, energy-awareness and environmental factors represent the major constraints in such applications. The first aspect is due to the possibility that the data can be launched and changed, e.g., during the transfer sessions. The second aspect is due to the use of battery-charged devices within the network. The third aspect deals with the noises influencing data transfer. The outlined factors are highly interrelated and extremely complex in their own way. For example, there are a variety of communication protocols designed to ensure different levels of security. The more complex a protocol is (in terms of the complexity of the encryption algorithms used), the more energy is required to ensure the required level of security. The same is true of environmental noises. All these factors, when considered together, predefine the quality of service (QoS) of the application. Therefore, QoS should appear as a basic non-functional requirement in designing the IoT applications. In [Jin et al., 2012], it is shown that QoS requirements for IoT applications are

very different even when the applications are very similar.

The IoT application itself consists of internal nodes (sensors, actuators, and other devices). Typically, the number of nodes ranges from a few to a dozen nodes. This depends on the type of each IoT application (e.g., a smart house requires fewer sensors than a smart city). The nodes may be combined into groups in order to cover different aspects of the same application, or even different but related applications. Therefore, the functionality and structure of a node can differ significantly, depending on the requirements of a particular application. On the other hand, there might be identical nodes (e.g., for ensuring better performance, higher reliability, etc.).

As the complexity of systems is steadily increasing, application development approaches should rely on the successful ones already in use in the industry, such as Product Line Engineering (PLE), Model Driven Application Development (MDAD) or Model-Driven Product Lines (MDPL). The PLE approach is defined as a methodology intended to develop a family of related products in an efficient way, taking full advantage of the products' commonality and variability aspects [Lockheed and U.S. Navy, 2013]. It is also concerned with the use of the variability management [Capilla et al., 2013]. MDAD raises the abstraction level of the typical application development by focusing on modelling and automated code generation from the models. Models of an application are specified by high-level feature modelling languages and using model transformations that typically are defined as model-to-program transformation [Czarnecki and Helsen, 2006] which are converted into the application. Therefore, the model-driven application development approach is centered on the use of models and their transformations [Czarnecki and Helsen, 2006]. MDPL combines the abstraction capability of MDAD and the variability management capability of PLE [Czarnecki et al., 2005a]. MDPL uses models with the intention to present the possible variability in order to implement the application in the given domain where the variability is usually modelled by using feature models [Dalgarno, 2007].

In IoT applications, due to their novelty and specificity, PLE approaches are not yet exploited in full to ensure better quality, higher productivity, more flexible adaptability and reuse despite the fact that IoT applications are domain-specific [Lopez et al., 2014] and share many common features in it. Therefore, the aim of the thesis is to analyse and disclose the potential of the product line engineering approach for designing IoT-oriented applications. Currently, the main problems in designing those applications are as follows: insufficient extent of automation, inadequate capabilities for process reuse, integration and adaptation. Furthermore, so far, designers have largely ignored the multiplicity and synergy of non-functional requirements (security and energy requirements, heterogeneity of devices and communication protocols) in designing their systems.

1.2. Object of the thesis

The object of this research is a feature modelling based method for the development of IoT-oriented applications product line.

1.3. Aim of the thesis

The aim of the thesis is the development of IoT-based applications in problematic domain(s) by using product line methodologies while accounting for IoT applications challenges: the complexity of requirements (security, energy consumption), the heterogeneity of technology, and the factor of the environment.

1.4. Tasks of the thesis

The main tasks of this thesis are:

1. To investigate the typical requirements and challenges of IoT applications development;

2. To propose an IoT applications product line development method based on feature modelling;

3. To perform transformations of feature models and specialized configuration to the framework of IoT application;

4. To produce practical implementation of the proposed method by implementing IoT application.

1.5. Scientific novelty

The thesis has achieved the following innovative results:

1. A novel feature model-based method for the development of IoT applications product lines has been proposed.

2. Generic functional and non-functional requirements feature models have been proposed which are used to present the variability of possible requirements of the IoT application in the application domain at an early stage of the application development.

3. The mapping procedure to map the problem domain variability presented by the feature model with the requirements of the specific IoT application has been proposed.

4. An approach to connect higher-level models with the generation level to produce IoT applications has been proposed.

1.6. Practical value

The creation of IoT-based applications is a very complex and time-consuming process because, during this process, software, hardware, the working modes of the hardware and the environmental factors must be combined in order to ensure the best performance (QoS) of an application in a given situation. Moreover, IoT applications are domain-specific and share many common features in the same domain [Chen et al., 2014], thus the product line technologies for the development of IoT applications can be used. PL methods allow to reuse the created application components for the development of another application. Considering that, IoT applications development methods with the following properties have been proposed:

- To present possible design space to implement IoT applications in a specific domain;
- To choose one implementation variant from the design space of some IoT application which meets the application's requirements optimally;
- To generate a framework of IoT application from the selected implementation variant.

The research results are included into project No. VP1-3.1-ŠMM- 08-K-01-018 "Research and Development of Internet Technologies and Their Infrastructure for Smart Environments of Things and Services" funded by the EU SA.

1.7. Thesis statements

1. IoT applications development methods based on feature models can be used to develop the product line of IoT application(s) in the specific IoT domain.
2. The generic feature models present: the complexity of requirements (security, energy consumption), the heterogeneity of technology, and the environmental factors of the problematic IoT domain.
3. The result of generic feature models transformations is a Pareto optimal model of IoT applications which is used to generate a framework of the IoT application.

1.8. Scientific approval

All of the results presented in the thesis are original; they have been presented in 4 internationally referred "ISI Web of Science" scientific journal publications and 2 other publications in informatics, electronics and software engineering peer reviewed journals and proceedings.

The experimental results were presented and discussed in 3 international conferences:

1. 18^{th} *International conference Electronics 2014*, Palanga;
2. 20^{th} *International conference on information and software technologies*, ICIST 2014, Druskininkai;
3. 12^{th} Annual *International Conference on Information Technology & Computer Science 2016*, Athens.

1.9. Thesis organization

The thesis consists of 5 chapters:

Chapter 1 is an introduction providing a short summary of the work's novelty, aim and objectives. This chapter includes a brief identification of the main problems in the IoT are and the motivation of the work.

Chapter 2 performs a thorough review of *Internet of Things*. This chapter describes in detail IoT applications, IoT applications development challenges, QoS of IoT applications and methods used for the development of IoT applications. The purpose of this chapter is to introduce the reader with the terminology and IoT applications characteristics and specifics which will be exploited and referred to in further chapters.

Chapter 3 presents a proposed feature model-based method intended to implement the IoT-based application. It starts with an overview of the proposed IoT application development method. The proposed method consists of five phases: development of models, aggregation, specialization, code generation and implementation. The processes of each phase are extensively described in a specific subsection.

Chapter 4 presents a prototype which implements the proposed IoT application development method presented in Chapter 3. It includes presentation of a case study of the implementation of IoT-based healthcare BAN application.

Chapter 5 is the concluding chapter where the proposed solution and contributions are summarized.

2. THE INTERNET OF THINGS

In this chapter, we represent the IoT since the understanding of the IoT serves as the background for this thesis. We shall discuss the definition of the IoT and review IoT applications, QoS of IoT applications and the methods used for IoT applications development.

2.1. Introduction to Internet of Things

The phrase 'Internet of things' (IoT) for the first time was mentioned in 1985 by Peter T. Lewis in a speech to the Congressional Black Caucus Foundation of the 15^{th} Annual Legislative Weekend in Washington [Saha et al., 2017]. Nowadays, IoT encompasses many aspects of life – from connected homes and cities to connected cars and roads to devices that track an individual's behaviour and use the data collection for 'push' services or even human healthcare systems which save people's lives. The goal of the IoT is to enable things to be connected 'anytime, anyplace, with anything and anyone ideally using any path/network and any service' [Smith, 2012]. Semantically, the IoT means a new highly heterogeneous world-wide network of interconnected objects uniquely addressable, based on standard communication protocols [Labiod et al., 2007].

There are several definitions of the Internet of Things; yet, these definitions vary among organizations and authors. In most of these definitions, IoT is presented as a dynamic complex system which combines various devices, where these devices are interconnected and share information with each other, and, for data sharing, they use standard communication protocols [ITU, 2005, INFSO, 2008, IERC, 2009].

We tend to think that IoT description presented by Gubbi et al. [Gubbi et al., 2013] best fits to the way how IoT is understandable in the scope of this thesis. The authors suggest that IoT is described as a smart environment framework, which interconnect sensors and actuators. Such interconnection provides the ability to share information across platforms through a unified framework. The proposed framework uses cloud computing for data analysis and information representation, which is generated by sensors and actuators.

Physical objects that interact with each other through communication protocols in the IoT context are usually understood as 'Things'. In this context 'Things'/physical objects could be electronic devices with network capabilities or any physical objects without communication ability.

Objects with communication capabilities are able to communicate with other devices by using similar communication capabilities. These devices usually communicate in Wi-Fi, Bluetooth and ZigBee wireless communication protocols (see Chapter 2.3). The number of this type of devices has increased dramatically. Nowadays, they are being used as everyday devices, such as smart phones, smart watches, refrigerators or TVs which are equipped with an embedded computer and one of connection protocols.

Objects without any communication capabilities can be used to present information about an object or to influence their state. These types of devices are usually presented by proxies which perform information transfer (devices or servers with communication capabilities) [Alam and Noll, 2010].

Despite the variety of types of objects which can be used in IoT, the sensing and actuating objects are the most important for IoT because sensing devices generate information about objects while actuating devices perform actions according to the collected data [ITU-T, 2012]. Most of IoT applications depend on the information generated by sensors because without this information applications cannot perform their tasks. IoT-based applications usually process the selected information and, according to the processed results, control actuating devices and provide other information.

Atzori et al. [Atzori et al., 2010] and Lee et al. [Lee et al., 2013] presented a survey of technologies that are often related to the IoT. The authors categorized the relevant technologies according to three perspectives which are understood as the IoT core technologies: things, network and semantic perspectives. In addition, these surveys state that IoT applications share many common features, such as technologies, communication protocols, data transmission, and the quality of service requirements.

2.2. IoT applications

Almost an infinite number of applications can be implemented by using the technologies of IoT. In the scope of the IoT domain, almost any types of application can be found, some of which are futuristic, while others are already being implemented and are used in the present days. The IERC [IERC, 2009, Vermesan et al., 2013] identified and described the main IoT applications which span across numerous application domains: smart energy, smart health, smart buildings, smart transport, smart industry and smart city. Chen et al. [Chen et al., 2014] distinguished nine domains of IoT applications which are very similar to the domain distinguished by IERC. The presented domains could be extended to many others, but almost every application derives from these main domains. These facts show that IoT applications are domain-specific and share many common features.

An IoT application can be understood as a centralized control which overlook connections between IoT application parts. The software side of the IoT application has to match and leverage the changes in the hardware. One of the well-known concepts, which helps to deal with the flexibility and reconfiguration requirements, is the service-oriented architecture. Due to the hard boundary condition, such as limited resources, the common concepts of services-oriented architecture cannot be directly mapped on most IoT applications.

Every application domain is denoted by its own characteristics. Most of these characteristics are presented, discussed, identified and analyzed in soft-

ware engineering literature; however, the IoT domain brings forward some characteristics which have not been analysed and discussed before. According to Patel et al. [Patel et al., 2011], IoT applications have the following characteristics:

- **Commonality at various levels.** Different IoT applications have a significant amount of features which are common between applications. It is caused by the fact that different applications are employing the same sensors or actuators, also, the same application can be deployed at different locations (e.g. the same patient's application at two hospitals) [Patel et al., 2013]. This shows that IoT applications are domain-specific.

- **Multiple concerns.** Commonality at various levels has a great impact on IoT applications for multiple concerns which must involve: 1) domain-specific features; 2) application-specific features; 3) operating system-specific features; 4) deployment-specific features.

- **Heterogeneous devices.** Usually, IoT applications must combine various sensing and actuating devices, and ensure communications and data exchange.

- **Heterogeneous platforms.** Different devices of IoT applications work and perform their actions by using heterogeneous platforms; the platforms are hardware-specific. For example, a device could be implemented by using a Smart-phone, Arduino, Blackberry, .net Gadgeteer and other hardware platforms. The hardware platform also determines the specific of operating system (e.g. Android, iOS, GNU/Linux, and others).

- **Heterogeneous interaction modes.** Different devices mean various methods how data can be accessed from them. Three data accessing modes can be distinguished: publish/subscribe [Fukui et al., 2013], request/response [Neufeld and Goldberg, 1990] and command [Andrews, 1991].

- **Scale.** Device network of an IoT application may consist of hundreds to million devices, which usually perform multiple actions.

- **Evolution.** An IoT application works in the environments, which change over time. Due to this fact, the IoT application must be modified in response to the changes; this means that some or all parts of the application should be re-developed. The changes may require adding or removing devices, adding new features etc.

2.3. Communication between 'Things' in IoT

Nowadays, most local area networks are based on the TCP/IP protocol [Hong-You and San-Ping, 2012] based wired or wireless communication networks. Wireless communication standards, such as Wi-Fi, and Personal Area networks, such as Bluetooth and ZigBee, are more frequently employed to facilitate the interconnections between mobile devices and the home appliance. However, the industry may use other communication protocols, such as Fieldbus [Shoshani et al., 2010] or CANBus [Huang et al., 2013]. Despite the heterogeneity of communication protocols, TCP/IP-based communication is the *de facto* standard on the Internet. Usually, IoT application tries to integrate the heterogeneous devices which communicate in different communication protocols. Moreover, these communication protocols must ensure different requirements for the data transfer rate and security. In order to show differences between various communication protocols, we looked closely at several wireless communication protocols which are used for IoT applications. Below, we will explore these protocols with respect to the data transfer rate, the level of security and the working range.

Bluetooth is a short-range radio frequency communication for exchanging data. Bluetooth technology is used in fixed and mobile devices, building personal area and IoT networks. Nowadays, almost all mobile devices have the Bluetooth technology. Communication between the devices is master-slave based [Davies, 2002]. The master controls all the communication, and the slave cannot communicate directly. A master can communicate either point-to-point or point-to-multipoint. A group of Bluetooth devices forms a cell called *piconet*; if several piconets overlap, they create a 'scatternet' network. The same Bluetooth device can belong to several piconets at the same time; thus it lets expand the coverage area of the Bluetooth network.

Bluetooth low energy or Bluetooth 4.0 [Honkanen et al., 2004], is a simplified version of the classic Bluetooth standard. BLE has a similar transmission rate and a slower data transfer rate but has a superior power saving capability and reduced time which is needed to connect to other devices. However, Bluetooth low energy single mode devices are not interoperable with classic devices such as Bluetooth 2.1+EDR devices. Single mode devices are only compatible with other Bluetooth low energy devices [Libelium, 2014].

Wi-Fi is a technology that allows electronic devices to exchange data or connect to the Internet by using radio waves [Khanduri and S. Rattan, 2013]. This technology is based on IEEE 802.11 standards for wireless local area networks. Many devices, ranging from smart-phones to sensors, have Wi-Fi connectivity. These devices can connect to a network or Internet by using a wireless network access point. In addition, wireless communication devices can be connected into an *ad-hoc* network, where devices communicate directly. Wireless devices communicate with each other by using unique identities.

Table 1. Comparison of Wi-Fi, Bluetooth, BLE, ZigBee, 6LoWPAN and Z-Wave protocols

Standard	Bluetooh	BLE	Wi-Fi	ZigBee	6LoWPAN	Z-Wave
IEEE spec.	802.15.1	802.15.1	802.11a/b /g	802.15.4	802.15.4	–
Max signal rate	3 Mb/s	1 Mb/s	54 Mb/s	250 Kb/s	250 Kb/s	40 Kb/s
Nominal range	10 m	10 m	100 m	100 m	100 m	100 m
Encryption	E0 stream cipher	AES stream cipher	RC4 strem cipher (WEP), AES block cipher	AES block cipher	AES block cipher	AES
Authentication	Shared key	Shared key	WPA2 (802.11i)	CBC-MAC	CBC-MAC	Key exchange
Data protection	16-bit CRC	24-bit CRC, 32-bit Message integrity check	32-bit CRC	16-bit CRC	CBC-MAC	CBC-MAC
Operation (security) mode	Mode1 (Unprotected), Mode2 (Encryption), Mode3 (Full encrypted)	Mode1 (Unprotected), Mode2 (Encryption), Mode3 (Full encrypted)	Unprotected, WEP-64, WEP-128, WPA-TKIP, WPA-AES, WPA2-TKIPT-AES, WPA2-AES	None, MIC-32, MIC-64, MIC-128, ENC, ENC-MIC-32, ENC-MIC-64, ENC-MIC-128	Non-secure, access control list, secure mode	Secure mode

ZigBee [Chen et al., 2006] ZigBee [Chen et al., 2006] is a standard for low-power wireless personal area networks (WPANs) and IoT systems, which is, in other words, wireless networks with a short range. ZigBee is built on top of the 802.15.4 specification which defines the physical and media access control layers. ZigBee can have one of the three different topologies: star, tree and mesh. Communication devices, according to the actions which they perform, are divided into three types: coordinating, router and the end device. Every ZigBee network must have a single coordinator which initialises the rest of the networking by defining the communication frequency and identifiers and allowing other devices to join the network. The router is responsible for relaying messages to other devices and is not required in all the network topologies. End devices are simple devices that send and receives messages.

6LoWPAN [Tabish et al., 2013] is a protocol which enables efficient use of IPv6 over low power, low rated wireless networks on simple embedded nodes. The target of this low power communication protocol is the applications that need wireless Internet at lower data transfer rate for devices with very limited resources. Such a connection can be performed by using border routers which connect the 6LoWPAN network with other IP networks.

Z-Wave [Yassein et al., 2016] is a wireless communication protocol oriented towards home automation. Z-Wave enables communication between devices directly or indirectly by using wireless mesh networking technology. In order to achieve this, Z-wave uses a network controller which controls all the communication in the network.

2.4. Energy issues within IoT

Energy consumption is the key problem in the wireless sensor network (WSN) as well as in the IoT. Communication is one of the most energy demanding tasks of IoT devices. In order to reduce energy consumption, low power communication protocols have been proposed from different standardization technologies, such as: ZigBee, Bluetooth low energy, RFID, and others (see more in Chapter 2.3).However, the communication characteristics in WSN and IoT are different from the traditional wireless communication because most of the devices use battery as the main power source. In addition, the number of the devices participating in communication is very large. Most of the time, IoT networks should operate for a long period of time without any need for human intervention [Kim et al., 2014]. Thus, energy saving is very important for WSN and IoT.

Abbas and Yoon [Abbas and Yoon, 2015] propose the taxonomy of energy saving issues for different types of wireless communication technologies of IoT. The presented taxonomy distinguishes among three main wireless communication technologies which are used in IoT: wireless area networks, wireless local area networks and wireless personal area networks. Each power saving method

of each main communication technology is analysed by using the following technical criteria: schemes, metrics, control, and evaluation.

In order to understand the energy consumption and saving problems in WSN and IoT better, some methods are proposed which suggest using device(s) modelling at an early stage of the wireless sensor and IoT network implementation. Zhou et al. [Zhou et al., 2011] describes the energy models of the WSN node core parts, such as processors, radio frequency modules and sensors. The basis of energy models is the event-trigger mechanism. The authors first simulate the node components and then estimate the energy consumption of network protocols while using these energy models. The model presented here is suitable for WSNs and IoT energy consumption analysis, for the evaluation of network protocols and for WSN or IoT application developments. Kamyabpour and Hoang [Kamyabpour and Hoang, 2010] propose a robust architecture that takes into account all the principal energy constituents of WSN applications. Their paper presents a single overall model and proposes a feasible formulation seeking to express the overall energy consumption of a generic WSN application in terms of its energy constituents. The formulation offers a concrete expression for evaluating the performance of a WSN application, optimizing its constituents' operations and designing more energy-efficient applications. Schmidt et al. [Schmidt et al., 2007] describes a method to construct models for sensor nodes based on few simple measurements. This source provides a sample where models are integrated in a simulation environment within the proposed runtime framework to support the model-driven design process. Measurements show that the proposed model enables to significantly reduce energy consumption.

Other authors investigate energy consumption during the communication process between WSN or IoT devices while using different communication protocols. This enables the prediction of communication device energy consumption in the implemented network and the application of the power saving methods. Lanzisera et al. [Lanzisera et al., 2014]proposes a 'communicating power supply' to enable the communication of energy and control information between the device and the building management system. According to the authors, the 'communicating power supply' technology can significantly reduce energy consumption in automated interactive solutions. Friedman and Krivolapov [Friedman et al., 2011] deals with a combined effect of power and throughput performance of the Bluetooth and Wi-Fi usage in smartphones. Their study discloses some interesting effects and trade-offs. In particular, the paper identifies many situations in which Wi-Fi is superior to Bluetooth thus countering previous reports. The study also identifies a couple of scenarios that are better handled by Bluetooth. The conclusions from this study give the preferred usage patterns that might be interesting to the researchers and smart phone developers.Venckauskas et al. [Venckauskas et al., 2014a] present the configurable IoT

prototype unit that enables to perform various experiments in order to determine the relationship between energy and security in various modes of the IoT unit. The paper also presents a methodology of measuring the energy of the IoT unit. While being applied, the methodology provides results in two different modes: ideal (without the effect of noises within a communication environment where the IoT unit works) and real (with the effect of noises).

2.5. Security and privacy challenges within IoT

The fact that most of IoT components are characterized as a low capability in terms of both energy and computing resources (this is exclusively the case for sensors and passive components) causes great concern of IoT security as well. As a result of its low capabilities, most of the contemporary cryptographic techniques are impossible to use or require further analysis. Due to this, most IoT devices do not protect the collected data and protect it only during the data sending operation of the encryption algorithm of the communication protocol. Because of these complications and the fact that the IoT network is usually deployed at a large scale, IoT is highly vulnerable.

Moreover, the heterogeneity of IoT networks increases the requirements for IoT security because different communication protocols and algorithms must be combined together in order to ensure the security and privacy requirements of IoT applications. Thus, it decreases the opportunity to implement IoT applications in a large scale [Roman et al., 2013]. Due to this fact, the security factor in the implementation of IoT applications is to support the scalability and heterogeneity of IoT applications. In [CoIoToTPC, 2013], data protection, privacy and information security are presented as complementary requirements of IoT services. In addition, Weber [Weber, 2010] considers new security and privacy challenges from the international legislation that pertains to the right to information, provisions prohibiting or otherwise limiting the use rules on IT-security legislation supporting the use of mechanisms of the IoT.

Chen et al. [Chen et al., 2009] delivers an overall vision of security issues in the sensor networks categorized as follows: cryptography, key management, attack detections and preventions, secure routing, secure location security, secure data fusion, and other security issues. The authors also summarize the techniques and methods used in these categories. Babar et al. [Babar et al., 2010] provides analysis of the IoT in the context of security, privacy, and confidentiality issues and proposes a security model for the IoT.

Some authors propose new methods while others investigate some opportunity to use the already existing methods to cope with the existing challenges of IoT security and privacy. Skarmeta et al. [Skarmeta et al., 2014] proposes a distributed capability-based access control mechanism. The latter is based on public key cryptography in order to cope with some security and privacy challenges in the IoT. Their solution uses the optimized elliptic curve digital

signature algorithm inside the smart object. Slavin et al. [Slavin et al., 2014] introduces security requirements patterns which represent reusable security practices that software engineers can apply in order to improve security in their systems. The paper proposes a new method that combines an inquiry-cycle based approach with the feature diagram notation intended to review only the relevant patterns and quickly select the most appropriate patterns for the situation. Heer et al. [Heer et al., 2011] discusses the problems and application possibilities of the already known Internet protocols and security solutions in the IoT. The authors also describe the deployment model, the core security requirements and emphasize the technical restrictions which are specific to the standard IP security protocols.

Despite the proposed new security and privacy methods, most contemporary IoT implementations are using the already existing standard methods. Therefore, method(s) are required which can analyse the presently existing security and privacy methods according to different requirements for security, privacy and energy consumption of each IoT implementation independently. In addition, security and privacy are contradictory to energy consumption because higher requirements of security and privacy increase the energy consumption, which poses the energy consumption problem of IoT applications (see Chapter 2.4).

2.6. Quality of service of IoT

Different parts of IoT application must communicate with each other, and this communication can be understood as services. According to [Sancho, 2009], a service is defined as an "asset of functions provided by a (server) software or system to client software or system, usually accessible through an application programming interface." Vermesan and Friess [Vermesan and Friess, 2014] presented a more accurate IoT application service description: "IoT application service is providing thing-related service within the constraints of things, such as privacy protection and semantic consistency between physical things and their associated virtual things." Services between different parts of IoT application(s) must perform the appropriate actions which are described by the quality of service in short QoS. A service in IoT can be defined by the combinations of capabilities of functionality, interoperability, interactions, communication abilities, related data and the ability of using the related data of the device(s) for implementing the IoT system in order to meet the requirements of specific application(s) and end user system(s) [Bhaddurgatte and Kumar, 2015]. As can be seen from a description of the quality of service of the IoT application, services describe only non-functional features of IoT applications

There is no formal definition of QoS; however, in the telecommunication domain, where QoS is mainly used, several definitions are available for certain communication level properties of the network. International Telecommunica-

tion Union (ITU) presented standard x.902 [Iec, 1995] where QoS is defined as "A set of quality requirements on the collective behaviour of one or more objects." QoS attributes describe the speed and reliability of data transfer.

In [ITU-T, 2005], the definition of QoS of IoT has been proposed, where QoS is defined as a complex indicator which evaluates user satisfaction with a given service. These indicators are related to security, user satisfaction, energy consumption, cloud computing provider parameters, and reliability [Han et al., 2013, Shaoshuai et al., 2011, Peng and Ruan, 2012, Hu, 2015].

QoS satisfaction level depends on many various parameters and attributes. The terms of the parameters and attributes are used interchangeably, and the set of QoS parameters for an IoT application and its environment defines the QoS provided by the application. Some QoS parameters and attributes have a bigger impact on user satisfaction of the IoT application. Therefore, methods are proposed which improve or optimize one or many QoS parameters. These parameters and attributes could be divided into four groups: 1) user and application perspective; 2) operator and network-specific; 3) edge nodes and communication system resource; 4) perspective of the all enabling technologies [Zheng et al., 2014, Bhaddurgatte and Kumar, 2015].

Teixeira et al. [Teixeira et al., 2011] reports that the IoT and the Internet of Services together will provide services for users in their real environment. The authors review the main problems faced by the service-oriented middleware along with the related scientific and technical innovation to support the services for the IoT. The authors distinguished among six main problems: scale, heterogeneity, unknown topology, unknown data-point availability, incomplete or inaccurate meta-data and conflict resolution.

- **Scale.** Usually, IoT application must control from several to millions sensing and actuating devices, which is a challenging task because each device must be coordinated, and these devices have different constraints such as latency, memory, processing power and energy consumption [de Fuentes et al., 2015].

- **Heterogeneity.** Different sensors and actuators of an IoT application can operate in different communication protocols. Also, they may have been manufactured by an assortment of vendors and can have different attributes which are related to the sensing performance, actuating capacities and accuracy [Mattern and Floerkemeier, 2010]. It results in IoT application heterogeneity because of the functionalities of the sensors and actuators and their capacity to communicate and exchange data among each other or with the systems exploiting them. All of these parameters lead to a rather challenging heterogeneity that makes the IoT extremely hard to work with.

- **Unknown topology.** Many IoT applications are denoted by unknown

main ability and dynamic topology. According to [Hachem et al., 2011], this leads to two consequences: 1) the application requires services which could be available when required; 2) services might themselves rely on the devices that had once joined the network and left it abruptly, or may require devices which never existed at the IoT application location in the first place.

- **Unknown data-point availability.** This problem appears because of the unknown topology. A situation may arise that there is no device which could accept the data and process it, and the device sending this information is not capable to collect/store the data.

- **Incomplete inaccurate meta-data.** Since much of meta-data must be manually entered by a human operator at installation time, in a massive network this will surely result in a large amount of incomplete/inaccurate information due to human error [Eisenhauer et al., 2010]. Moreover, some meta-data information changes over time.

- **Conflict resolution**. Conflict resolution appears when several applications try to control the same devices. This problem usually affects actuators [Hachem et al., 2011].

2.6.1. QoS evaluation and optimization

The QoS for the same IoT application can be composed of various components (e.g. different devices, communication protocols, security requirements) which have their own parameters and attributes; this opportunity to select the different options creates a set of possible variants to implement the QoS. These different variants of implementation must be evaluated in order to meet the individual needs of many applications. However, sometimes, evaluation is not an option to select the best composition of parameters to implement the QoS because evaluation produces a big number of possible compositions. In this case, the optimization is used; at this point, QoS optimization means finding an optimized composition with respect to several QoS constraints (e.g. minimizing the power consumption while maximizing the security level), which is the best for the specific IoT application. Depending on the specific end requirements of the IoT application, optimization criteria could be related to energy, network bandwidth, query, routing efficiency, network coverage, and other aspects.

Peng and Ruan [Peng and Ruan, 2012] proposed a QoS service model which is based on analytical hierarchy process. This model takes user preferences and device performance parameters, and evaluates this information in order to select the best QoS for users.

Shaoshuai et al.[Shaoshuai et al., 2011] proposes the multi-objective decision-making by using the evaluation model of service quality. This model

takes into consideration both the state of the system and the user settings to improve the model of the QoS validity. The calculated assessment of the proposed model can be used as a parameter for the estimation and selection of the service.

Table 2. Multi-dimension QoS parameters of IoT applications

Dimension	QoS parameters		
Sensing	Accuracy	Availability	Stability
Transmission	Transmission time	Storage capacity	Reliability
Application	Functionality	Normative	Robustness

Li et al. [Li et al., 2014] presents the method for evaluating and optimization of the QoS of IoT. The proposed method evaluates IoT application by using multi-dimensional QoS. QoS of an IoT application is understood as a space of n dimensions, where each dimension presents one of the parameter of QoS. These requirements cope with the heterogeneity of IoT applications because applications cover various sensing and actuating devices which can communicate in different communication protocols. Table 2, presents the proposed multi-dimensioned QoS parameters of IoT applications.

In [Li et al., 2015], the authors presented the methodology which evaluates the users' satisfaction degree of the IoT application according to the optimized quality of the service of IoT application. The proposed methodology evaluates the IoT quality of experience by using multiple linear analysis. First of all, the type of the IoT application must be analysed, the quality of the service parameters for the quality of experience is determined, and the principal components are distinguished. Then, the regression function between the quality of experience and the principal components is composed; after that, the functional model between the quality of experience and the QoS parameters is created. Such an evolution of QoS helps reduce the complexity of the quality of experience and the number of parameters of QoS.

Guan et al. [Guan et al., 2006] presented the framework for QoS-guided service optimization while using constraint hierarchies as a formalism for specifying QoS. In the proposed framework, a model of QoS is created from functional requirements which are modelled as hard constraints and use constraint hierarchies; however, this framework supports a small amount set of QoS attributes.

2.7. Methods of IoT applications development

In this chapter, we provide a comprehensive survey of the state of the art in the IoT applications development. The main purpose of this chapter is to investigate the already existing approaches.

2.7.1. MDD and IoT applications

In [France and Rumpe, 2007], France and Rumpe say that one of the challenges in the application, including IoT, development is filling the gap between the application domain and the technology used to implement the solutions of the problem. Two different concepts: the domain space and the technology domain (computer science domain), must be understandable in order to create a solution to the problem domain. The domain space represents the concepts such as information flows, the behaviour and process with the scope of the application domain. After that, the domain space must be mapped with the technology domain (the computer science domain), which represents programming languages, memory and storage management as well as communication technology. Thus, application engineers must possess strong analytic skills and creativity because they must combine both domains into one in order to create the solution application [MacDonald et al., 2005]. Furthermore, Greenfield and Short [Greenfield and Short, 2003] claim that this combination of the two domains cannot be done by using mass production approaches.

Model driven development (MDD) is an applications development approach which is used to build the bridge between the domain space and the technology domain. Developing an application by using model-driven development implies the creation of a model of the problematic application and the automatically generated code [Nikiforova et al., 2009]. Application domain modelling helps reduce the risks of bad engineering and usually decreases the production time of the application. Application domain model creation is understood as domain analysis [Czarnecki and Helsen, 2006], and this step of application creation is performed at the highest abstraction level because modelling languages used in the creation of models, are of higher abstraction level than the application implementation language. Moreover, Sallai [Sallai, 2014] identifies modelling as a separate branch in the IoT research.

The lowest abstraction layer of MDD methodology is code generation from the created application domain model [Fuentes et al., 2009]. In the generation process, modelling languages are getting transformed into machine-executable codes. This process is done by code generators or compilers.

In [Riedel et al., 2010], the authors presented a flexible code generation method based on model drivel development for the creation of an IoT-based application. The proposed method strives to build the bridge between the networked embedded objects and the enterprise back-end system. In order to do that, modelling and code generation architecture has been proposed; this architecture is based on state of the art technologies which ensure efficient data exchange between two different parts of the IoT application. For modelling, they use various high-level message description languages in order to create visual automata which are used as the basis for the code generator.

In [Grace et al., 2016], the authors presented an MDD-based method to

control the interoperability of IoT application development. This is necessary because the IoT resource is highly heterogeneous in terms of communication technologies, protocols and data formats. Due to this, the authors proposed to use two types of the model: the interoperability model and the specification models. Thus it allows to reduce the IoT application development time and to ensure possibilities to create complex applications. Small models designed to describe the interoperability are used; this allows faster simulation and evaluation of IoT application.

In [Patel and Cassou, 2015, Patel et al., 2015], the authors presented an MDD-based IoT application development framework which copes with IoT application problems, such as heterogeneity, the large number of devices, and the different life cycle phase. In the proposed development framework, a methodology that separates IoT application development into different concerns, and the methodology which allows to support the actions of stakeholders are suggested. The framework provides a graphical modelling language which allows to specify each development concern and abstracts, and integrates code generation, task-mapping and linking techniques in order to provide automation. Code generation is done in such a way that it supports all the IoT application development phases. Furthermore, the framework is oriented in such terms as re-usability: the extent to which software artefacts can be reused during application development; and expressiveness: the characteristics of IoT applications that can be modelled while using this approach. Conceptual models have been chosen as the main modelling technology.

As can be seen, most of the proposed MDD methods for IoT applications development are oriented to model 1 application. However, in [Chen et al., 2014], it is stated that IoT applications will be deployed in nine domains: smart house, smart medical care, intelligent transportation, smart agriculture, smart environmental protection, smart safety, smart grid, domain industry application and smart logistics. In addition, according to Internet of Things Environment for Service Creation and Testing [Lopez et al., 2014] project, the already existing IoT applications are domain-specific because they use heterogeneous communications, technologies and protocols. These facts show that IoT applications share a lot of common features in the same domain; thus the product line methodologies can be used for the creation of IoT applications in the same domain. These techniques are used to present the possible ways to implement an IoT application in its domain at an early stage of its development, and to ensure good re-usability of components (models, software fragments, and others) while developing different applications in the same domain.

2.7.2. Modelling languages

Model-driven development requires language(s), with which, the domain model can be created. There are various forms of modelling languages, such as graphical notation, hierarchical tree and textual languages. All these languages can be divided into two main types: programming and modelling [Poruban et al., 2014]. By using a programming language, the application domain is presented in the form of the source code, and as a model in case of modelling languages, which is transformed into the application source code. The similarities and differences of programming languages and modelling languages as model-driven development technologies are discussed in [Sun et al., 2008]. The authors noted that modelling languages mostly use higher abstraction, particularly, while using graphical notation. Meanwhile, programming languages are executable at a lower abstraction level.

Depending on the abstraction level which is presented by the modelling languages, and the representations, modelling languages are divided into two main categories: general modelling languages and domain-specific languages (DSL). General modelling languages can be used to describe any type of the application domain. Examples of general languages may be UML [Vanderperren and Mueller, 2006], or they may feature the model script language [Acher et al., 2013]. UML language uses graphical notation to describe the domain. By using UML architecture, objects, interactions between objects, data aspects modelling, as well as the design aspects of component-based development are involved. Feature model scrip language is used to present different variants in order to develop an application in its domain.

According to Mernik et al. [Mernik et al., 2005]: "Domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general purpose programming languages in their domain of application." As it can be seen from this definition, general purpose languages are more flexible than DSLs. A good example of DSLs is VHSIC Hardware Description Language (VHDL) [Swamy et al., 1995]: it is a standard DSL for describing digital circuit designs, Embedded Systems Language (ESML)[Jouault and Bezivin, 2006]: it is a standard DSL for describing digital circuit designs, Embedded Systems Language (ESML) [Jouault et al., 2008]: it is a DSL for specifying model transformations (more about DSLs can be found in [Sun et al., 2008]).

Due to the heterogeneity, huge scale, network requirements and other specific factors of IoT applications, most of the software engineering domain-specific languages cannot be used for the creation process of IoT applications. As a result of this fact, new DSLs have been proposed which are oriented to IoT applications. By using these DSLs, most aspects (e.g. heterogeneous objects) of an IoT application can be described.

Midgar is one of such languages which is used to describe smart objects

for various IoT application platforms [García et al., 2014]. By using Midgar DSL, the necessary software for objects is created. A graphical editor is used to present this DSL; also, this editor allows the user to create any kind of objects (sensors, actuators, etc.) which can be used in their platform (e.g. smartphone, micro-controllers). This helps to cope with the huge amount of heterogeneous devices. Moreover, a system developer can describe which data will be sent to the platform for further processing.

In [Salihbegovic et al., 2015], DSL-4-IoT domain-specific language has been proposed. DSL-4-IoT can be used to describe the following aspects of IoT application: heterogeneity of wireless sensor networks, devices, communication media, protocols and operating systems. This lets IoT application developers describe the most specific IoT applications. The editor of this language is based on the class of visual domain-specific modelling languages. DSL-4-IoT uses formal presentations and abstract syntax in a meta-model. The hierarchical structure is as follows: the system, subsystem, device physical or virtual channel is used to describe IoT application by using DSL-4-IoT.

2.8. Product line and IoT applications

The traditional focus of applications development [Acher, 2011] (including IoT applications) and the analysed MMD methods are intended to develop individual applications. As it was mentioned earlier, almost all the IoT applications can be divided into nine main domains [Chen et al., 2014], where these domains can be divided into smaller IoT application domains. Due to this fact, IoT applications can be seen as a product group which has a common set of features: similar developing platforms and implementation functionality in the same domain. Thus the product line (PL) methodologies can be used for IoT applications development in order to show as many as possible variants to implement the IoT application in its domain. PL engineering relies on the idea of mass customization [Kotha and Pine, 1994] known from many industries: telecommunications, avionics and others. Such an application development method tries to identify whatever applications may have in common and manage the aspects varying among them while analysing the problematic domain. The main purpose of PL engineering is to provide various development-related solutions for different customers in a systematic and coordinated way. Instead of individually developing each variant from scratch, commonalities are conceived only once. Therefore, the final product is customized to meet the specifications of individual customers.

PL ensures better reusability of components between different applications because a group of applications share common features in the same domain [Glass, 2001]. Therefore, the common components (features) can be reused between different applications in a group, and the application can be developed from the beginning instead of one by one from scratch [Lee et al., 2002].

According to [Fuentes et al., 2009], PL brings two new issues as compared to the engineering of single product-based systems: the variability design and the product derivation. The variability design is responsible for the variation mechanisms incorporation in the production process of products in such a way that it would be possible to present an infrastructure to describe the complete range of family products. Such an infrastructure is used in order to describe both the commonalities and the variations of the family of products. Product derivation is the process of a specific product creation from the selected specific configuration by selecting it from the configurations which were determined at the variability design step. In order to perform variability design, various variability modelling techniques were introduced (see Chapter 2.8.1).

The development of IoT applications has many features in common, but we must also estimate a high variability due to the heterogeneity of the communication protocols, devices, and environment facts. Therefore, demand appears to solve component management issues between different applications of the IoT with respect to the inherent variability of these applications. Due to this fact, suggestions are made to use product line technologies for IoT application(s) development.

Anon et al. [Anon et al., 2014] presented an IoT application development framework which manages variability and provides an opportunity to create IoT applications. In order to achieve that, they adopted the product line engineering and proposed a framework with a layered architecture which consists of a cloud layer, a central hub layer, and an end devices layer. The proposed framework allows to present the variability of all layers.

Conejar and Kim [Conejar and Kim, 2016] presented a framework for the mobile device product line security based on the Internet of Things which is used for the development of IoT application in mobile devices. This framework uses PL engineering in order to present variability at the following three levels: edge, access, and application. According to the authors, this improves the planning, operations and control phases of the product development.

Ayala et al. [Ayala et al., 2015] proposes a method which uses product line engineering for the development of IoT agents by using Self-StarMASMAS, a multi-agent system. The proposed method can be divided into three steps: IoT multi-agent domain analysis, IoT multi-agent system variability modelling and IoT multi-agent system architecture. The first step is used to collect information about the IoT multi-agent domain with respect to the devices and transport protocols heterogeneity which can be used in the creation of agent in the IoT applications. The variability model of the analysed multi-agent domain is created in the second step. During the third step, the final application architecture is automatically derived from the variability model. These three steps ensure an appropriate reuse mechanism to develop an agent family of applications in the IoT.

All of the analysed PL methods for IoT applications development focused on modelling a variability (configurations) to implement the application in its domain. However, none of the analysed PL methods considers the security levels of the communication protocols, the device working modes and the device energy consumption factors as a part of IoT applications development, which have an impact on an IoT application's performance (QoS). Despite this fact, these factors are crucially important for the development of IoT applications.

2.8.1. Variability modelling

The main task of presenting the variability of the PL is to distinguish the features shared by all the products of the line, and the elements that may vary from one product to another. Many different techniques are proposed for the presentation of variability among products in the same domain. In [Berger et al., 2013], twelve main variability notations are distinguished: feature model, spreadsheet, key/value pairs, domain-specific language, UML-based representation, decision model, free-text description, product matrix, aspect-oriented language, architecture description language, configuration facilities of a component framework, and goal model. Bellow, we present several variability notions which use models so that to present the variability in PL.

Feature models for the first time were proposed by Kang et al. [Kang et al., 1990] in 1990. Feature models are usually standard for representing variability in PL. Despite years of research, there is no universal modelling standard for feature models. Due to this fact, a number of proposed extensions exist. Most of the proposed extensions are based on parent and child relationships. Feature models are presented as a hierarchical diagram to organize a potentially large number of concepts. These hierarchical diagrams (see Appendix B) are usually graphically presented as rectangles while some graphical elements, such as a filled or unfilled circle are used to describe the feature type 'mandatory' or 'optional' (see Appendix B), and the root feature is the most general concept. The original feature model notation called FODA was proposed by Kang et al. [Kang et al., 1990]. Griss et al. [Griss et al., 1998] presented the Feature-RSEB (based on Reuse-Driven Software Engineering Business) feature model as a FODA extension with additional relationship. FODA feature models use four types of relations among the features to present the PL variability: mandatory, optional, alternative, and cross-tree constraints: require and exclude. Later on, Feature-RSEB feature models extended the FODA notation by adding OR-relation between the features. Riebisch et al. [Riebisch et al., 2004] and Czarnecki et al. [Czarnecki et al., 2005b] proposed cardinality-based feature models where cardinality was introduced as the main concept when striving to present the variability in the PL. These three types of feature models are most common in literature.

The UML is one of the most popular notations for modelling and ap-

plication by proposing a set of models intended to specify several aspects of the application. However, it is unsuitable for modelling groups of related applications as required by PL [Ziadi and Jézéquel, 2006]. Therefore, the UML extensions to present variability were presented. In [ClauB and Jena, 2001] three UML extensions are presented in order to demonstrate the variability. The first extension describes the rules how to transform feature models into UML models. The second extension is used to present the variation points of feature models. The third extension intends to present an optional element of the feature model. The final two extensions were implemented by using stereotypes. Ziadi and Jézéquell [Ziadi and Jézéquel, 2006] presented another UML extension which tries to present the variability in PL engineering as it is possible by using feature models (FODA notation). In order to achieve that, they proposed three stereotypes: optional, variation and variant. In addition, the meta-level constraints of the object constraints language to present constraints, which are used in FODA feature models, were presented as well. Maazoun et al. [Maazoun et al., 2016] presented an UML profile named SPL-UML, which models variability in UML designs for PL. The profile uses the proposed stereotypes in order to extract information from the feature models and presents it as class diagrams, sequence diagrams, and uses cases diagrams.

Haugen [Haugen, 2012] presented a common variability language which is used to express the variability between models in terms of the variation point (model fragments, such as placement fragments) and variants (replacement fragments) [Echeverria et al., 2015]. The creation of product models is perfumed by changing model fragment(s) between the base model (placements) and a model library (replacements).

According to [Lee et al., 2002, Berger et al., 2013] both industry and academia use variability modelling in their PL for the creation of a wide range of applications domain. Moreover, industry and academia use feature model notations as the main methodology for variability modelling. Due to the wide use of feature models, there are many tools, both commercial and open source, which support feature models as a variability modelling notation (despite the product domain whose variability is presented) and provide opportunities for analysing the created variability models in various aspects. In addition, according to [Czarnecki et al., 2012], feature models allow to model both the commonality and variability of PL, thus enabling to define the product derivation from the PL. These facts show that feature modelling is a domain independent notation; therefore, it can be used in the development of IoT applications in the problematic domain by presenting an application's PL in the domain of interest.

2.8.2. Model transformation

Both model driven development and product line techniques use model transformations in order to create the desired application from the cre-

ated high-level models. Model transformations allow transforming one model to another automatically. The classification of model transformation was proposed by Czarnecki and Helsen in [Czarnecki and Helsen, 2006, Czarnecki and Helsen, 2003]. The authors presented three categories of transformation:

- Text-to-model or code-to-model. This category comprises parsing and reverse-engineering technologies.
- Model-to-code. The target of this transformation is the programming code.
- Model-to-model. This transformation produces its target as an instance of the target model.

Text-to-model and Code-to-model transformation is a reverse engineering process where the model is generated from the source code by using some tools (from code to higher-level representations). Thus the model is described by using the same existing programming language in order to create a high-level model. This transformation is usually described as model-driven reverse engineering and is used to simplify the production of the new version of the system thus reducing the risk of mistakes [Rugaber and Stirewalt, 2004]. Object Management Group presented a knowledge discovery model which is used to express the various types of knowledge extracted from a legacy system and covers not only the structure of the code but also data modelling, event handling and other concerns [Garcia-Dominguez and Kolovos, 2016].

Model-to-code transformation is usually referred to as a code generation process, which transforms high-level models to programming code. Model-to-code transformations are described by rules and pattern-matching like the activation schemes of these rules. Such a transformation is widely used in programming code generators. In order to perform model-to-code transformation, usually, a number of code generation templates are used for code generation from the model(s) [Kalnins et al., 2010]. Each template holds some particular aspect code, which is linked with the corresponding parts of the model(s) during the transformation process.

The process of converting one model into another is called model-to-model transformation. The model transformation here means the mapping of the two models (or more) based on transformation rules or update of the same model. It is important to distinguish between 'endogenous' transformations (e.g. models merging and re-factoring), in which, the source and the target models are the same, and 'exogenous' transformations (e.g. code generation from a model), where the two models are different.

2.8.3. Code Generation

Code generation can be understood as a model transformation process. According to [Thibault and Consel, 1997], application generation is a process when a target application is created from a high-level specification (model). Generator programs transform abstract data (model) into a code making it possible to manage the software at a favourable abstraction level of the model [Stuikys and Damasevicius, 2013]. As input for a generator, various types of files, such as the application model, intermediate code, and others can be forwarded. In other words, code generation is used to produce programs automatically. In addition, the use of high-level specifications in code generation increases a model's re-usability in different applications. One of the code generation methods is used by MDD and PL methodologies.

The code generator can be divided into several types, which depend on how they treat the source code. Three code generator types are distinguished: template-based, model-transformation mechanisms, and meta-programming with reflection.

The model-transformation generators use the application model as input; this model is called the application meta-model, and is created by using any modelling language including any type of domain-specific language. The meta-model can be treated as the grammar of a modelling language. Such a code generator uses various model transformation approaches [Czarnecki and Helsen, 2003, Hemel et al., 2009] for code generation. Model transformation approaches describe the relation between the source and target models, which depends on the application. The relations show what should be transformed into what.

Meta-programming generators manipulate the other program in order to generate the output program; reflection is a good example of such manipulation, and compiler is a good example of meta-programming generators. Meta-programming usually uses an abstract syntax tree, which is usually written in a meta-language to generate a code [Miao and Siek, 2014]. While using this kind of generators, the code can be generated as specialized for different cases (programs) [Nanevski, 2002]. However, the meta-programming code generators are limited, because they can generate structural constructs like classes, methods, attributes.

Template-based code generation approach relies on the use of templates which denote the way to transform the input data of the generator into textual files [Franky and Pavlich-Mariscal, 2012]. Templates are pre-compiled code fragments; these fragments include all the text which will be included in the output file. Most templates are platform-dependent because they offer a solution for a certain combination of hardware, operating systems and programming language. This fact enables to re-use these templates in different implementations using the same platforms. Thus template-based code generation is a good

approach for code generation of IoT applications because, as it was mentioned above, IoT applications share many common features including platforms in the domain.

2.9. Conclusions

This chapter is aimed at reviewing the challenges and development approaches used in the development of IoT applications. The literature analysis defined the following issues of the IoT application development process:

1. During the IoT application development process, the heterogeneity of communication protocols, sensors, actuators and other devices must be analysed and adapted to any IoT application separately even in the same domain.

2. There is lack of IoT applications development approaches which analyse and model the possible variability seeking to implement IoT applications in the application's domain despite the fact that IoT applications are domain-specific and share many common features in the same domain.

3. Due to the fact that most IoT devices have a low capability in terms of both energy and computing resources, the security and energy consumption must be analysed as main components to ensure the necessary QoS of IoT during the application development process.

4. There is lack of IoT applications development approaches which combine IoT application requirements for heterogeneity, security and energy consumption and analyse them according to QoS requirements of IoT application at an early stage of application development.

5. In order to solve the problems of IoT application development which were distinguished during the analysis of IoT applications development process, product line methodologies can be used.

3. FEATURE MODEL-BASED DEVELOPMENT OF IoT APPLICATIONS

Analysis of the IoT domain shows that most IoT applications have a very similar structure, which, with very few changes, can be adapted to the majority of implementations of IoT applications. The basic structure of an IoT application is shown in Figure 3.1 [Gubbi et al., 2013, Vermesan et al., 2013, Vermesan and Friess, 2014, Chen et al., 2014, Zachariah et al., 2015]. This structure consists of the following components: measurement-communication module (MCM), standard Internet module (SIM) and IoT application data collection module (IoTADCM).

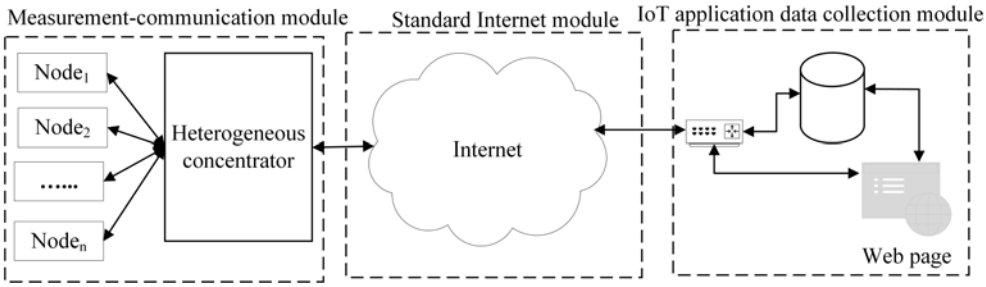


Figure 3.1. The basic components of IoT application

MCM has two main tasks: to generate data and to send it to IoTADCM via the SIM or to perform some actions according to the control signal which can be sent from IoTADCM. Commonly, MCM is the core of IoT application because this module is responsible for data generation (e.g. temperature, humidity, and others) and action performance; according to collected data, these two actions are 'vitally' important for the functionality of the IoT application. As it can be seen from Figure 3.1, the measurement-communication module consists of two parts: the nodes network and the heterogeneous concentrator (HC). IoT nodes (see Explanation 1) network may be composed of different hardware components which can collect contrasting measurements and perform various actions and communicate in different communication protocols thus creating the heterogeneous network, usually of low range. Nodes perform data collection; if necessary, they also perform data processing, and send it to the concentrator. While devices performing actions wait for a controlling signal from the concentrator, these signals usually come from IoTADCM and sometimes can be generated by the concentrator. HC is usually a much bigger and more powerful device compared with the remaining components of the measurement-communication module, and it commonly has unlimited power supply. HC collects data from IoT nodes, performs data processing and prepares data for transferring to the IoT application data collection module via the SIM; it can also send control

data to IoT nodes which are derived from the IoTADCM. The concentrator is necessary because, most of the time, IoT nodes use the low range communication protocols which usually cannot send data via the standard Internet or network(s) directly. Therefore, HC processes MCM collected data so that it could be sent via the standard Internet to the IoTADCM which can actually be thousands of kilometres away from the measurement-communication module.

Explanation 1 *IoT node – IoT application unit, which consists of an action module and a communication module.*

Explanation 2 *Action module – IoT application hardware component (sensor, actuator, or other), which performs actions (measures, controls, or others).*

At this point, the standard Internet module represents the Internet as it is understood today. The global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) links billions of devices worldwide. The Internet consists of millions of private, public, academic and other networks which are interconnected.

The IoT application data collection module is responsible for collecting, storing and final processing of data which was collected by MCM. If the IoT application requires that, the module performs decision taking when decision taking can be related to the control of MCM devices; most of the decisions are based on the processed data which came from MCM. In Figure 3.1, it is shown that this module can also be responsible for the presentation of data to the end user (e.g. a web page or other graphical user interface technologies).

Such a viewpoint of the three modules at the IoT applications gives an opportunity to use product line methodologies for the creation of a measurement-communication module of IoT applications as this module of IoT applications is consistent with many features which are common for almost all the IoT applications within the same domain. Thus, PL methodologies help to ensure the component re-usability between different applications and to reduce the development time of the IoT application because an application can be created by selecting components from the existing component libraries where the components can be both hardware and software. However, there are many problems in the IoT domain because the same components must ensure very different requirements for security, energy, data transfer rate (e.g. the same temperature sensor used in agriculture and healthcare will be implemented with the same software components, while the same or different communication protocols can be used for data transmission). Naturally, problems develop how to evaluate these requirements and choose the best components from the component library because a lot of factors have an impact on those requirements. The evaluation of these factors affecting the selection of components and the component working modes can be understood as the quality of service (QoS) measurement for the IoT-based application.

Due to these facts, we proposed an IoT application development method which uses feature models (Feature-RSEB notation which is presented in Appendix B) to present the variability of configurations (different devices, communication protocols, ant other) to present the variability of configurations (different devices, communication protocols, and others) to implement the IoT application in a specific domain, evaluates each configuration according to the QoS requirements and suggests the best configuration to implement the specific IoT application. Feature models were selected on the grounds of the following facts: a) these models are widely used to present the variability of product lines in various application domains; b) are independent from the application domain c) have many software tools which support feature models. In the scope of Chapters 3 and 4, the configuration is understood as a list of hardware components, communication protocols, data transmission information, and security parameters that can be used for the development of an IoT application.

3.1. Proposed IoT application development method

Based on the above-mentioned possibility to use product line methodologies in creation of IoT-based applications, the five phases of IoT applications development method were proposed. They cover the architectural design and implementation software development stage, both at the domain and application engineering levels. The first two phases are used for the engineering of an IoT application's domain and serve for the presentation of the variability of possible configurations (different PLs) to implement the application. The last three phases correspond to the creation of the specific IoT application by: 1) selecting the configuration (hardware components and their working mode) which meet the application QoS requirements best; 2) generating code framework of the IoT application from reusable software components for the selected hardware; 3) implementing the IoT application. Figure 3.2 presents the phases of the proposed IoT applications development method. The five-phase method was proposed in order to simplify domain analysis of the IoT application and to ensure better re-usability of the created IoT application domain models and software components in the creation of various IoT applications.

Development of domain models. The purpose of this phase (at the requirements specification phase) is first to extract the relevant knowledge and then to represent it adequately so that it would be possible to apply the knowledge in the subsequent layers as easily and effectively as possible. Typically, the resulting knowledge of modelling is a set of models (e.g. functional and non-functional) of the domain under consideration. The modelling procedure, in order to be systematic and most useful for the next phases, requires the use of some well-defined approach. As stated above, currently, the feature-based modelling approaches prevail [Apel and Kästner, 2009, Almeida et al., 2015] where the design of new

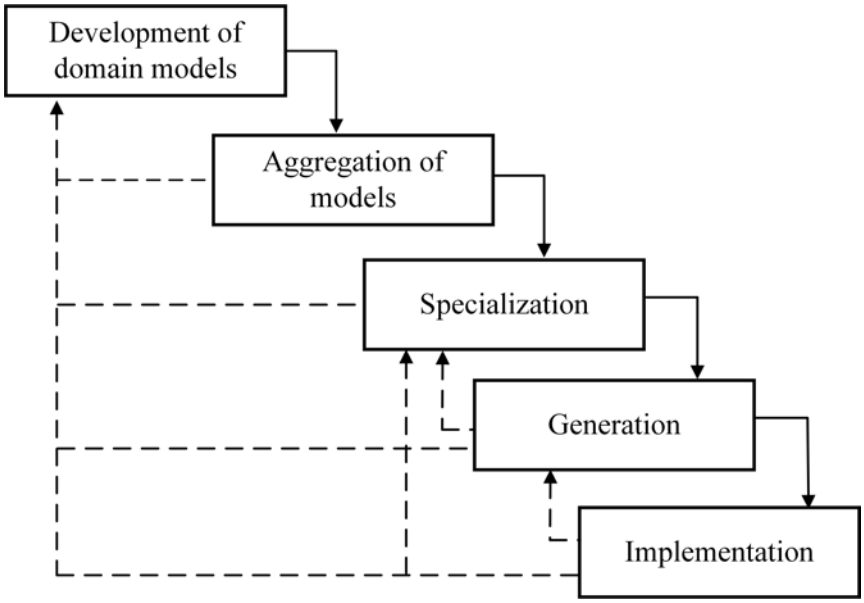


Figure 3.2. Method of the implementation of the IoT application

systems with the reuse in mind is considered.

Aggregation. The aggregation phase serves for aggregating the input of a set of models (e.g. functional and non-functional requirements models) into the resulting model. The main intention is to maximize the reuse potential in using the product line paradigm. The conditions of aggregating are as follows: (i) input models must be correct; (ii) they must be represented uniformly using the feature-based notion; (iii) the models are to be general enough (meaning the adequate scope of the domain expressed by features) to support the pre-defined extent of reuse. Aggregation might be carried out not for all the input models but for the specific separately selected models. For example, aggregation of a non-functional model (i.e., security and energy-related) are the most likely in terms of composition for the resulting model. The reusable model resulting from aggregation would still be an abstract one. The models should be correct. The consistency should be verified.

Specialization. Specialization can be viewed as the process of transforming abstract feature models into the concrete model. By the concrete model, we mean the one whose abstract features are decomposed into variant points, the latter containing variants as concrete values. Here again, one can see the processes, the actions performed by the designer as well as the tools to support the modelling and verification processes along with the produced products, i.e. feature-based models. We should note that only

specialized models can be useful at the generation phase. One important aspect of the specialization is that the application is not homogeneous and may contain different functional and non-functional aspects which can be understood as QoS. These functional aspects, when implemented, are represented by the different components, and these components must be selected according to the requirements and constraints of QoS. Thus, the specialization phase also includes these issues.

Generation. The generation phase is responsible for the production of the application code through the configuration-to-program transformations. This type of transformation is necessary for lowering the level of abstraction. This phase uses the best possible implementation of the application which was suggested by the specialization phase. As the next step, software components (communication protocols, data transmission, etc.) are selected within the specified specialized configuration; they will be used for the implementation of the application.

Implementation. This is the final phase of the proposed IoT application implementation method which is dedicated to creating the final working IoT application. It means that, at this phase, all the components (sensors, actuator, communicating and transferring facilities) of the IoT application are combined into one functioning application according to the predefined requirements and constraints. These actions require intervention of system developers who extend the generated framework code by applying additional functionalities which had not been generated in the generation phase.

3.2. Relationship between phases of proposed method

Figure 3.3 shows the relationship between the proposed method phases and represents the dependencies between them. As it can be seen from Figure 3.3, every next phase depends on the previous one because it uses the previous phase data as input for further actions.

Every activity which is done following the proposed IoT-based application development method starts from the analysis of the IoT application domain and applications in this domain. ***In the scope of Chapter 3, the measurement-communication module of IoT-based application will be understandable as an IoT domain (see Figure 3.1)***. During the analysis of the IoT domain, the functional and the non-functional requirements feature models are created. These two models should present as many as possible configurations of functional and non-functional requirements so that to implement application in the analysed domain. In short, the output of the model development phase is two models: functional and non-functional requirements in the graphical or

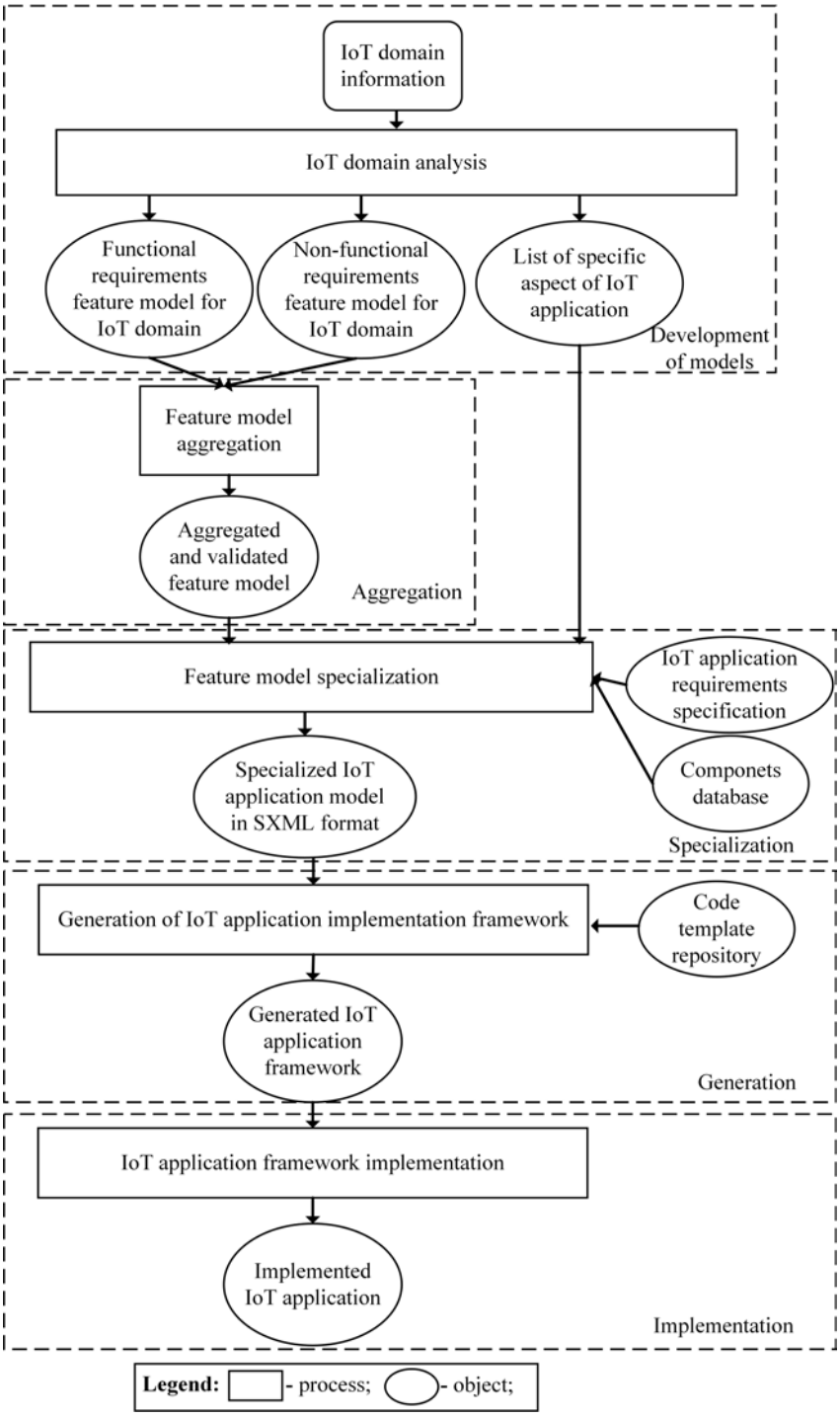


Figure 3.3. Relationship between the method phases

textual or both formats as it depends on the tools which are being used for the creation of the feature models.

The aggregation phase is responsible for the creation of one resulting model which combines both functional and non-functional requirements feature models and their constraints. This means that it takes the created functional and non-functional models and aggregates them. Aggregation takes two (or more) feature models and creates one resulting feature model which must be correct. At this point, the correct feature model must be seen as a model, which does not have any dead features (see Definition 21 in Appendix A), in other words, the features and constraints of the resulting model are compatible. The correctness of a newly created feature model is ensured by application developer(s) by using automated tools. The output of the aggregation phase is an aggregated and verified feature model of analysed IoT domain and which presents many possible configurations (the variability of PLs) which can be used for the implementation of IoT applications in the analysed domain.

The specialization phase creates a specialized feature model of the IoT-based application and selects the best configuration to implement the application. The specialized feature model describes all the possible correct configurations (PLs variability) which can be used for the implementation of the specific application. In order to create a specialized feature model, IoT-based application specifications are merged with the aggregated feature model. The created specialized feature model usually describes several configurations; thus one configuration that meets the IoT application's requirements best must be selected. The selected configuration is used to generate a framework of the IoT-based application. The selected configuration describes hardware components which should be used for different nodes implementation in order to implement the specific IoT application. The described hardware components are as follows: sensors, actuators or other components and communication modules. The selected configuration also presents the working mode which must be used for each communication module. In addition, for the selection of the best configuration to implement the IoT application, each configuration is evaluated according to the IoT application's QoS requirements. The output of the specialization phase is the specialized configuration in the XML format, from which, the framework of the IoT-based application will be generated.

The generation phase is responsible for the code framework generation from the specialized configuration of the specific IoT application. Code generation is performed by using a code template repository [Ge and Whitehead Jr., 2008]. The generation phase takes configuration as an input, parses it, and determines what source code components (methods) must be selected from the code template repository in order to generate the framework of the IoT application. Components are selected for all the modules which are included in the specialized configuration file. The generated framework de-

scribes only the initial functionality rather that the fully working application. A fully working application is obtained in the implementation phase by supplementing the generated framework with the necessary functionality.

The implementation phase takes the generated code framework of a specific IoT application as an input, and complements it till the working application by adding the necessary functionality and logical connections between the generated code parts.

All the actions and transformations which are performed by the proposed method can be described by referring to the following steps:

Step 1 *Abstract functional and non-functional requirements feature models are obtained through domain analysis (DA) and modelling by using the relevant DA approaches (such as FODA [Kang et al., 1990]) along with adequate tools.*

Step 2 *If an abstract model consists of separate models, then model aggregation follows. The latter merges two or more input models without common features into the resulting model (aggregated, see Definition 22 in Appendix A) by using the tools.*

Step 3 *An aggregated feature model is transformed into a specialized one by extending some lower-level features into features with concrete values so as to satisfy the design aims and requirements of IoT application.*

Step 4 *A specialized model is derived from the aggregated model under given specific requirements and specialization rules (such as narrowing the number of configurations).*

Step 5 *The aggregated problem domain feature model is transformed into the specialized solution feature model by using the aggregated feature model through mapping of the corresponding IoT application specifications (e.g., variation points and variants) onto the features of the specialized problem domain feature model.*

Step 6 *The variation points of the specialized solution feature model correspond to the parameters within the IoT application framework which will be generated, and the variants of the variation point correspond to the parameter value.*

Step 7 *The specialized solution model is transformed into a specialized configuration by using design space exploration [Mahalank et al., 2016] through mapping of the corresponding specialized solution model values with the components contained in the components database.*

Step 8 *The specialized configuration is transformed into the IoT application framework by performing the following actions: 1) selecting the source code*

related to specialized configuration values; 2) creating a framework from the selected source code whose structure depends on the programming language specifications.

Step 9 *The IoT application framework is transformed into the fully functional IoT application via the following actions: 1) adding the necessary functionality and logical connections; 2) testing the framework functionalities.*

3.3. Development of domain models

This phase can be understood as an analysis of the IoT application domain. Domain analysis includes the methodologies and processes used to create and manage the product line. The FODA [Kang et al., 1998] acts as a domain process to collect the domain knowledge, study the product family structure, and create a reference architecture for the product family of applications. FODA describes the basic concepts of modelling as an abstraction and a refinement [Kang et al., 1990]. The process of studying the existing applications, generalizing the functionalities and designs into generic domain elements is called abstraction. Refinement is the process of creating a specific application from the generic domain elements by adding factors that are unique for individual applications.

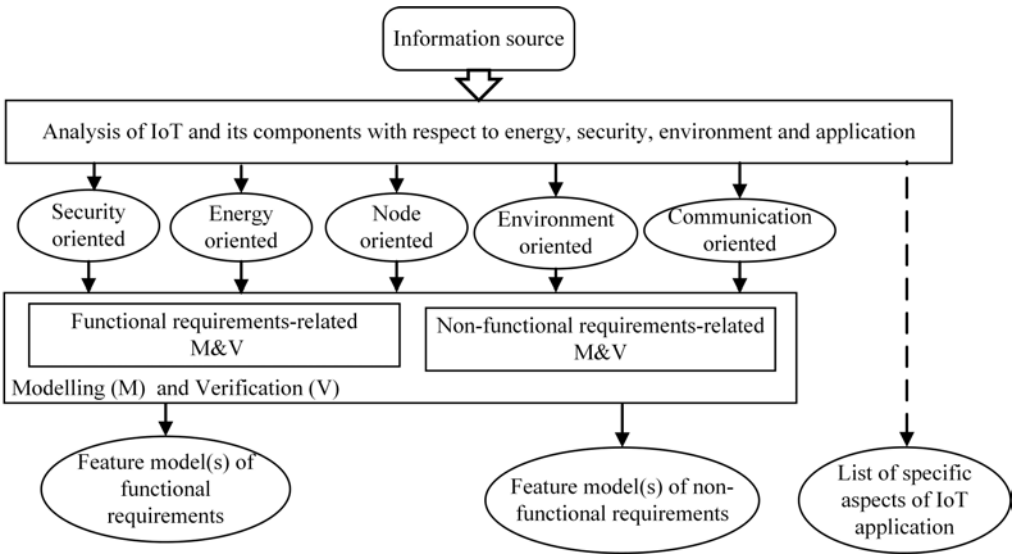


Figure 3.4. Models development of IoT application domain

In the proposed IoT domain models development phase, only an abstraction concept is used from the FODA modelling concepts. The abstraction concept is used to create feature models of the functional and non-functional requirements of an IoT application domain. As it can be seen from Figure

3.4, feature models are created from the collected data at the analysis process of the IoT domain with respect to application, energy, security, environment, nodes and communication. Experts extracting the most important data which properly describes the application functional and non-functional requirements in the problematic domain of the relevant IoT application perform the analysis. The main task of this phase is to present as many as possible configurations (product line) of the functional and non-functional requirements which can be used for the development of IoT applications in the analysed domain.

Considering the feature definition as "a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems" [Kang et al., 1990] and the generic structure of the IoT application (see Figure 3.1), we propose the generic structure of the functional and non-functional requirements feature models. We believe that the proposed models exhibit a good opportunity to be reused and fitted for various developments of IoT-based applications. In addition, these feature models cover all the areas of interests which were distinguished in Figure 3.4 and are used to present the variability of possible configurations (PLs) to implement the application in the domain. The proposed models are shown in Figure 3.6. These models are not final and can be extended according to the experts' needs. However, in our opinion, the mandatory features (see Definition 5 in Appendix A) of the proposed feature models must stay as they are shown in Figures 3.5 and 3.6. Thus only the lowest level of the proposed feature models should be adapted in order to present configurations (PLs) of the functional and non-functional requirements of the IoT application in its domain.

However, there are some exceptions, security level, power consumption and physical obstacle features of non-functional requirements models. We suggest that these features should be presented as it is shown in Figure 3.6a for any IoT domain, whereas models of IoT application domain are created. This means that no changes or additional features are required. Hence we believe that the presented lowest-level features are sufficient to describe the most important aspects of the IoT application domain. Due to this fact, IoT domain modelling can be performed faster.

We presume that [Venckauskas et al., 2014a, Venckauskas et al., 2014b, Venckauskas et al., 2016a] shows that the presented six levels of security (see [Pastore and Dulaney, 2006]) are sufficient to describe any protection which is required by an application and can be provided by communication protocols. These six levels of security can be used to describe communication security requirements of any IoT application. The same holds with the features describing the environment, in which the application of the IoT domain usually operates. It is covered by the three proposed features. The power consumption for various configurations of the IoT domain is sufficiently covered by the three features as well. Furthermore, the security level, the power consumption and the physi-

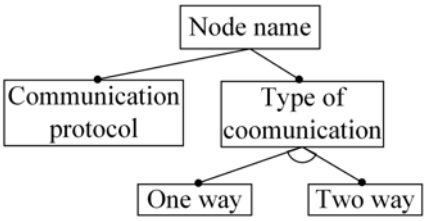


Figure 3.5. Feature model fragment which represents the IoT node

cal obstacle features are used to show the security-energy, environment-energy, security-environment-energy relationships which are important for the application's performance. These relationships are also used for the selection of a configuration, from which, the IoT application framework will be generated at the generation phase. The configuration selection is performed by calculating the QoS values of each configuration presented by a specialized feature model (see Chapter 3.5).

The proposed structure of functional and non-functional requirements feature models can be extended by supplying additional features which can be optional or mandatory and can have XOR or OR (see Definitions 8 and 9 in Appendix A) feature groups. However, the basic structure of the feature models, as it is shown in Figure 3.6, must stay unchanged. This means that new changes should not change the mandatory feature names. Consequently, the work of the following phases is based on this recommended structure of functional and non-functional requirements feature models of the IoT application domain.

In addition, we suggest that the children of the *Node* variant point (see Definition 7 in Appendix A) should be presented by using the structure which is shown in Figures 3.5 and 3.6b. Such a structure is used because the Node feature children describe an IoT node (see Explanation 1), where an action module performs various actions which are important for the IoT application whereas the communication module is responsible for data transferring in and out of the action module. Due to these facts, such a structure of the feature model striving to describe the IoT node has been proposed.

Furthermore, an extensive or extremely complex domain can be divided into smaller sub-domains in order to perform the analysis easier. Thus, for each sub-domain, functional and non-functional requirements models should be created. These models would be combined into one model in the aggregation phase of the proposed IoT application development method (see Chapter 3.4).

At the level of domain models, the development phase of the proposed IoT application development method, and the feature models should present as many as possible configurations (the variability of PLs) so that to implement the functional and non-functional requirements of the application, most of which

will be eliminated at the specialization phase. Furthermore, feature models presented in Figure 3.6 are shown without constraints because they depend on the IoT domain of application; however, feature models, at this point, must be understood as a combination of the feature model and its constraints (see Definition 1 in Appendix A)

Feature models presented in the proposed generic functional and non-functional requirements models were distinguished during the analysis of the most important facts for IoT applications and their QoS [Atzori et al., 2010, Lee et al., 2013, Patel et al., 2011, Teixeira et al., 2011, Zheng et al., 2014, Bhaddurgatte and Kumar, 2015]. The proposed non-functional requirements feature model presents the following features which were distinguished during the analysis:

Security level. It describes the IoT-based application requirements of security for data transmission (communication). Each security level describes the specific working mode of the communication protocol; these working modes are presented in [Venckauskas et al., 2014b, Venckauskas et al., 2016a].

Energy consumption. This feature is used to show different energy requirements of different configurations where it can depend on communication protocols, IoT devices, security levels.

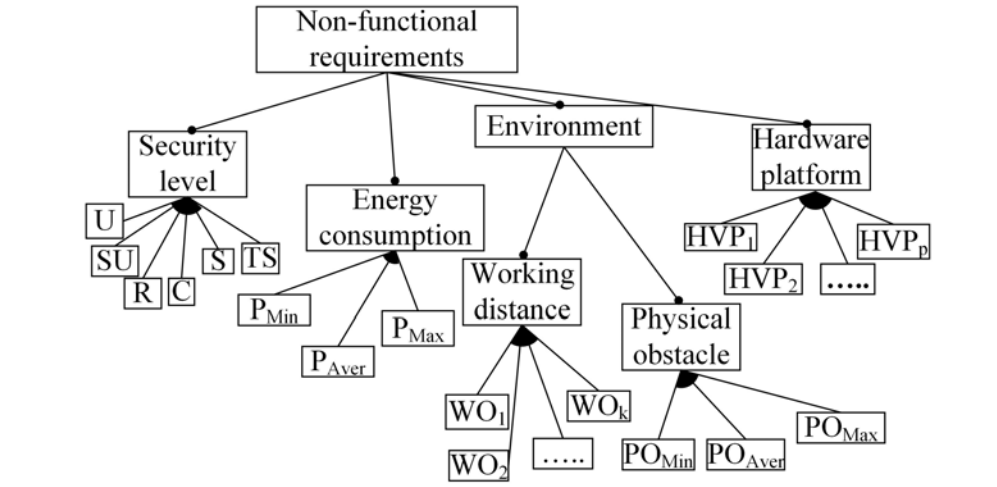
Environment. It describes the environment in which the IoT application is expected to work. In Figure 3.6a, the environment feature is divided into two features: working distance and physical obstacle.

Working distance. It presents the distance within which IoT nodes must operate. In other words, it shows how far one IoT node is from another node or device which collects the data and controls it.

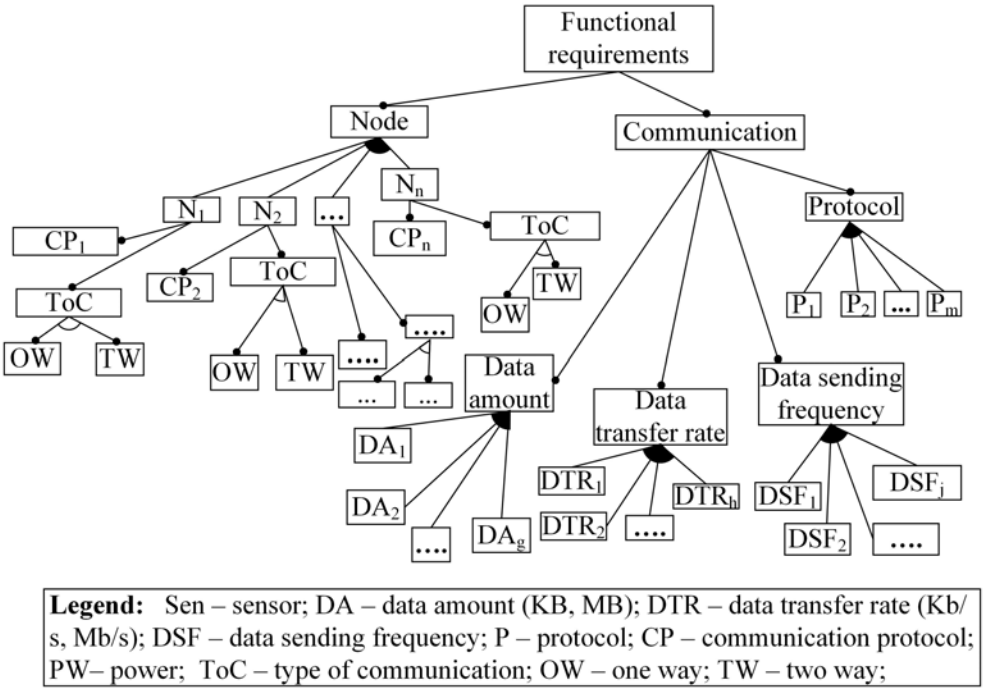
Physical obstacle. It describes how many obstacles which can interfere with the communication signal are in the working area of the IoT application. The obstacles are understood as an environmental factor which influences the transfer of wireless communication signals. We distinguish among three values: minimum for the rural regions, the maximum for city regions and the medium level falling into the space between the extreme values.

Hardware platform. It describes hardware platforms whose components are or can be used to implement the analyzed IoT-based application (e.g. Arduino, Gadgeteer and others).

The proposed functional requirements feature model contains the following features:



(a) Feature model of non-functional requirements



(b) Feature model of functional requirements

Figure 3.6. Generic structure of the proposed feature models

Node. It describes the possible nodes which can be or are actually used in the analysed IoT domain.

Communication protocol. It describes the communication protocols in which action modules can operate. Each individual action module usually operates in one protocol.

Type of communication. It describes the types of communication in which the action module communicates with a heterogeneous concentrator and other IoT nodes; the communication type depends on the action module type. As it can be seen from Figures 3.5 and 3.6b, we distinguish between two types of communication: one-way and two-way communication. *One-way* describes the communication where action modules can only send data. *Two-way* describes the communication where action modules can send and receive data.

Communication. This feature is used to show the most suitable information about the communication properties of the analysed IoT applications in their domain.

Data amount. It describes how much data is usually sent in one communication session by action modules.

Data sending frequency. It describes how often data is usually transferred to other IoT devices by the action module.

Data transfer rate. It describes what data transfer rate usually needs to be ensured by the communication module so that all the data generated by an action module could be transferred.

Protocol. It describes all the possible protocols which can be or are used for the implementations of IoT-based applications in the analysed IoT domain.

After the extension and adaptation of the proposed functional and non-functional feature models, we obtain feature models which completely describe the analyzed IoT application's domain (i.e. they present the variability of PLs of the functional and non-functional aspects of IoT application in their domain). As it can be seen from Figure 3.4, the newly created feature models must be verified. Verification is used to ensure that the created feature models are correct (i.e. they do not have dead features, see Definition 14 in Appendix A), and only verified feature models are used in the next phase of the proposed IoT application development method. The application developer performs verification by using model verification tools that perform verification automatically.

From Figure 3.4 it can be seen, that besides the functional and non-functional requirements imposed on feature models at the modelling phase, we

also suggest creating a list of specific aspects of IoT-based applications in the analysed domain. The specific information can be anything that cannot be presented in feature models or which could be used for the augmentation of IoT application specifications, and could be used for the specialization of the aggregated feature model (see Chapter 3.5).

3.4. Aggregation phase

As it was mentioned above, the aggregation phase serves for aggregation by creating the resulting feature model of the analysed IoT application domain. Aggregation takes two (or more) feature models (see Figure 3.7) and creates one. The main aim of aggregation is the interrelation and separation of feature models through cross-tree constraints. Features in input feature models are correlated to each other through relations expressions ('requires' and 'excludes', see Definitions 11 and 12 in Appendix A) [Streitferdt et al., 2003].

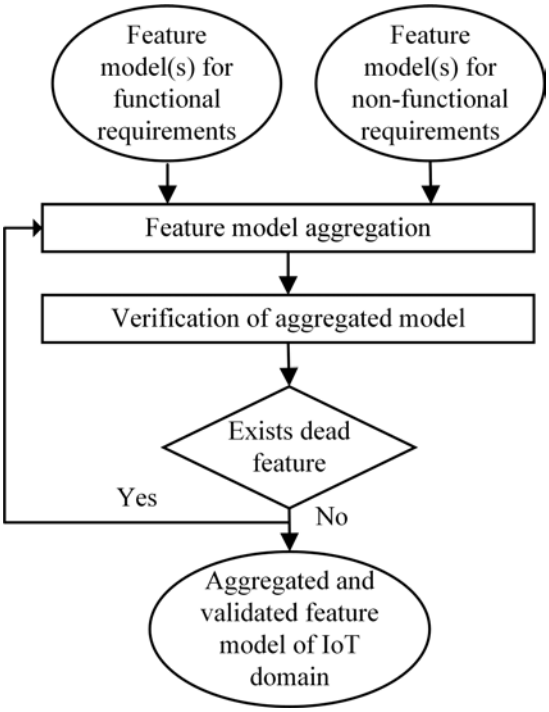


Figure 3.7. Aggregation layer

During the aggregation process, feature models are treated as equally important. Because of this, the distinction between models is not needed, to define the syntax or the semantic properties of the aggregation process. The aggregation process can be described as follows [Acher et al., 2010a]:

$aggregation(sIoTFM : setofIoTfeaturemodel, sIoTCons : setofconstraints)$

The aggregation process takes a set of feature models ($sIoTFM$) and and a set of constraints ($sIoTCons$) as input, and produces a new feature model which presents possible configurations (variability of PLs) in order to implement the IoT application in its domain. Input feature models are aggregated by creating a new root ($Root_{master}$) feature (see Definition 3 in Appendix A) so that the root of the input feature models is a child of the created new $Root_{master}$ root and requires mandatory connection.

In order to present why the aggregated feature model needs new root, we will use FM_{a1} and FM_{a2} feature models and a_1 and a_2 features. The new root is necessary because feature model FM_{a1} can describe the concepts that may not be composed of or refined by the concept described by feature model FM_{a2}, and help avoid the situation when feature a_1 of feature model FM_{a1} can be a root feature of the aggregated feature model while feature a_2 of model FM_{a2} becomes its child with the mandatory connection. It follows that two models may be unconnected, and the relation between FM_{a1} and FM_{a2} should be reversed.

After aggregation, we have an aggregated feature model with a set of new constraints which comes from FM_{a1} and FM_{a2} feature models. These constraints may a have different impact on the aggregated feature model. Aggregating two feature models FM_{a1} and FM_{a2} with a different set of constraints can create three resulting feature models, as follows: (for example, for our purpose, we used feature models presented in Figure 3.8) [Acher et al., 2010a]:

Reduced model. The reduced aggregated model is represented in Figure 3.8a, and can be disclosed as:

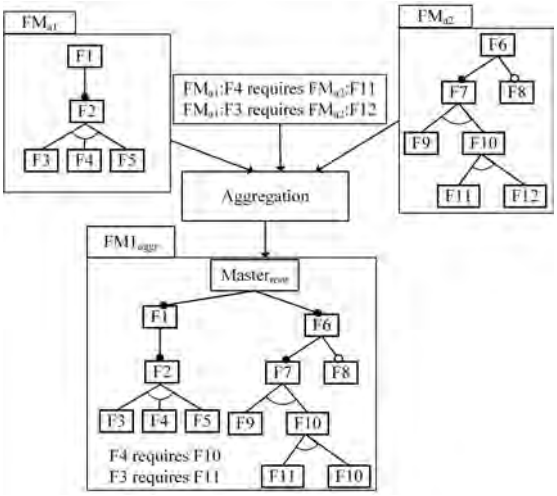
$$FM1_{aggr} \subset (FM_{a1} \otimes FM_{a2}), \tag{1}$$

by aggregation, FM_{a1} and FM_{a2} are combined together to create new $FM1_{aggr}$ which has new configurations. $FM1_{aggr}$ model cannot represent all combinations of configuration of FM_{a1} and FM_{a2} because of the constraints.

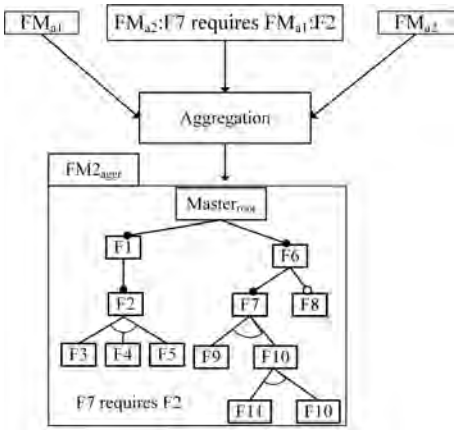
Redundant model. The redundant aggregated model is shown in Figure 3.8b, and can be disclosed as:

$$(FM_{a1} \otimes FM_{a2}) = FM2_{aggr}, \tag{2}$$

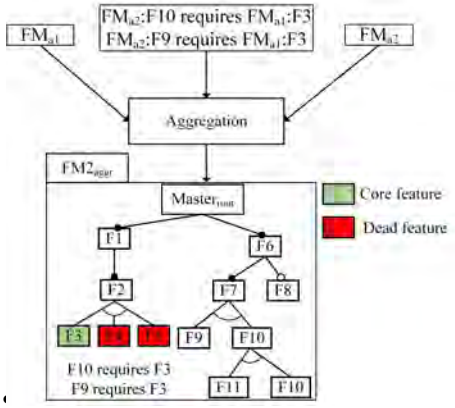
model $FM2_{aggr}$ does not reduce the set of configurations, and all combinations of configurations of FM_{a1} and FM_{a2} are allowed. It means that the aggregation of FM_{a1} and FM_{a2} creates an aggregated model which is equivalent to model of Figure 3.8a without constraints. Naturally, the fact that constraints *F7* require *F2* is logically entailed by the aggregation of FM_{a1} and FM_{a2} without



(a) Basic aggregation



(b) Redundant constraints



(c) Dead and core features

Figure 3.8. Three types of aggregation

constraints. This follows from the fact that these two features are mandatory and must be included in every possible combination of configurations of the aggregated model.

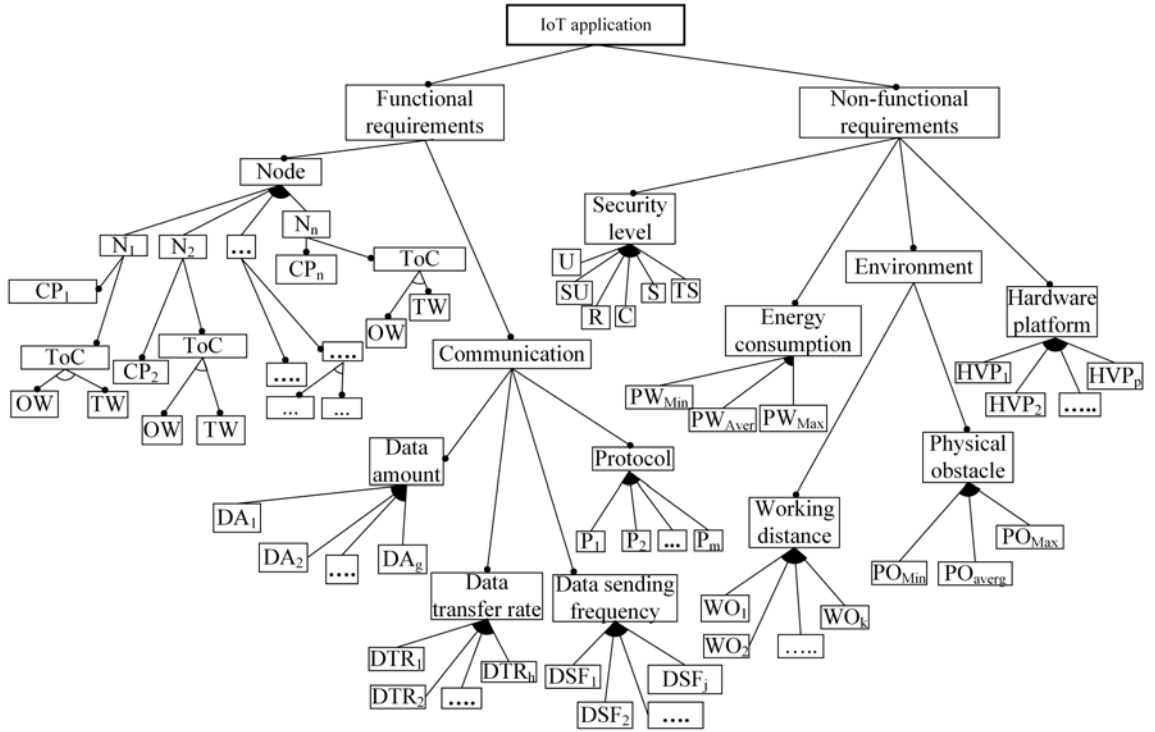
Dead and core features. The dead and core features aggregated model is shown in Figure Figure 3.8c, and can be disclosed as:

$$(FM_{a1} \otimes FM_{a2}) \subset FM3_{aggr}, \tag{3}$$

due to constraints, features *F4* and *F5* are now dead features (Definition 14 in Appendix A) while feature *F3* is a core feature (Definition 13 in Appendix A) of $FM3_{aggr}$.

After the aggregation process of functional and non-functional requirements for feature models, a new feature model (see Figure 3.9) is created which has a new root feature. The new root feature of the created aggregated feature model is suggested to be named according to the IoT application's name (e.g. Greenhouse, Food supply, etc.) whose analysis was performed at the model development phase. For example, the purpose, the root feature of the aggregated feature model, was called *IoT application* (see Figure 3.9). After aggregation, we get a reduced feature model because not all configurations of functional and non-functional feature models can be presented by it. Furthermore, a new list of constraints has been created, and this new list usually contains more constraints than functional and non-functional requirements models together. The new constraints appear from the fact that features of functional and non-functional requirements feature models can require (include) or exclude (see Definitions 11 and 12 in Appendix A) each other. These new constraints come from domain analysis. For example, Table 1 can be used as a good source for the creation of new constraints, which was not possible before aggregation.

By using our proposed functional (Figure 3.6b) and non-functional (Figure 3.6a) requirements feature models, it is impossible to get the redundant resulting feature model. Because, naturally, some constraints appear between communication protocols and the level of security or distance as well as the sensor operating distance as a result of restraining of communication protocols or hardware which were presented in the functional and non-functional requirements models. The result of addition of at least one new constraint between non-mandatory features into aggregated feature model is configuration(s) loss in functional and non-functional requirements feature model(s). Therefore, such newly created feature model does not comply the requirements of redundant feature model.



Legend: Sen – sensor; DA – data amount (KB, MB); DTR – data transfer rate (15kb/s, Mb/s); DSF – data sending frequency; P – protocol; OP – operation distance ; CP – communication protocol; HVP – hardware platform; PW– power; PO – physical obstacle; WD – working distance; ToC – type of communication; OW – one way; TW – two way; WO – working distance;

Constraints:
TS requires P_1;
TS and S excludes P_2;
WO_1 requires P_n;
P_n requires PW_2;
……..

Figure 3.9. Aggregated and verified feature model of IoT application domain

A set of constraints of functional and non-functional requirements feature models used during the aggregation or new constraint(s) can noticeably change the properties of the aggregated feature model. It may lead to situations where the aggregated feature model does not represent any valid configuration or may include dead or core features. The verification of the aggregated model is used to avoid such unacceptable situations. If there is a situation that, after verification, at least one dead feature remains, we must return to the aggregation process (see Figure 3.7). At the aggregation process, after carefully reviewing all the constraints, sometimes a dead feature can be fixed by deleting or adding constraint(s) such that all constraints would be compatible with each other, and that the aggregated feature model would be correct (see Definition 21 in Appendix A). Automatic model verification can be performed by using various software tools which support feature models.

If everything has been done correctly, we should get an aggregated and verified feature model, which combines functional and non-functional requirements feature models which have been created at the previous phase of the proposed method. Figure 3.9 presents the basic structure of the aggregated feature model which was created from the proposed functional (Figure 3.6b) and non-functional (Figure 3.6a) requirements feature models. As it can be seen from Figure 3.9, the created aggregated feature model contains all the features and constraints of functional and non-functional feature models. For the purpose of demonstration, we only display several possible constraints as a separate list due to the readability.

The main purpose of an aggregated feature model is to present as many as possible configurations which can be used to implement IoT application(s) in its domain. Thus an aggregated feature model usually presents a significant number of configurations (the variability of possible PLs to implement application in its domain). Some subsets of these configurations are not relevant for IoT application which we want to implement as the created aggregated feature model describes all the domain(s) of the analysed IoT application. This fact allows reusing the created aggregated feature model for the development of other applications in the same IoT domain. Thus the creation of an application can be started from the specialized phase of the proposed IoT applications implementation method. In order to reduce the number of configurations which are relevant for the IoT application being developed, the specialization of the aggregated feature model must be performed.

3.5. Specialization phase

After the aggregation phase, an aggregated feature model has been created. The created model presents the variability of configurations (PLs) which can be used for the implementation of application(s) in the analyzed domain. At the specialization phase, the aggregated feature models must be merged with

the specifications and the list of specific aspects (if it was created at the model development phase) of the IoT application which must be developed. In order to achieve that, the merging process is used. The merging process is used in order to decrease the number of configurations (the variability of PLs) which are presented by the aggregated model and are the most relevant to the IoT application which must be developed. The specialized model which presents these configurations is a result of the merging process. In order to achieve the consistency of the specialized model, new constraints may be added, some features may be deleted and some feature groups may change their form. In order to select one configuration from the specialized model which meets the requirements of the specific IoT application best, the design space exploration process is used. The specific IoT application's framework will be generated from the selected configuration in the generation phase.

According to Brunet et al. [Brunet et al., 2006], model merging is traditionally used to group together model elements that describe the same concepts in the input models to be composed. Syntactically, the merging operation can be defined as [Acher et al., 2010b]:

$$merge(sFM : setofFeaturemodels, mode : MergeMode)$$

In our case, the aggregated feature model, the specifications and a list of specific aspects (if it was created at the model development phase) of the specific IoT application are used as inputs. During the merging process, features of the aggregated model and specifications of the IoT application are merged (see Figure 3.10). During the merging process, the features and specifications which contain the same or synonymous information are merged.

Thum et al.[Thum et al., 2009] identifies and classifies four merging modes of feature models where $\otimes$ is a merging operator:

Re-factoring. After re-factoring, two or more models deliver model FM_{result} which is created and contains all configurations which were described by the models before re-factoring. This means that no new configurations are added and no existing configuration is removed.

$$FM_1 \otimes FM_2 = FM_{result} \tag{4}$$

Specialization. Specialization removes some existing configurations and does not add a new configuration.

$$FM_{result} \subset FM_1 \otimes FM_2 \tag{5}$$

Generalization. When merging two or more models by using generalization, new configurations are added to the resulting model, and no existing configuration is removed from the resulting model.

$$FM_1 \otimes FM_2 \subset FM_{result} \tag{6}$$

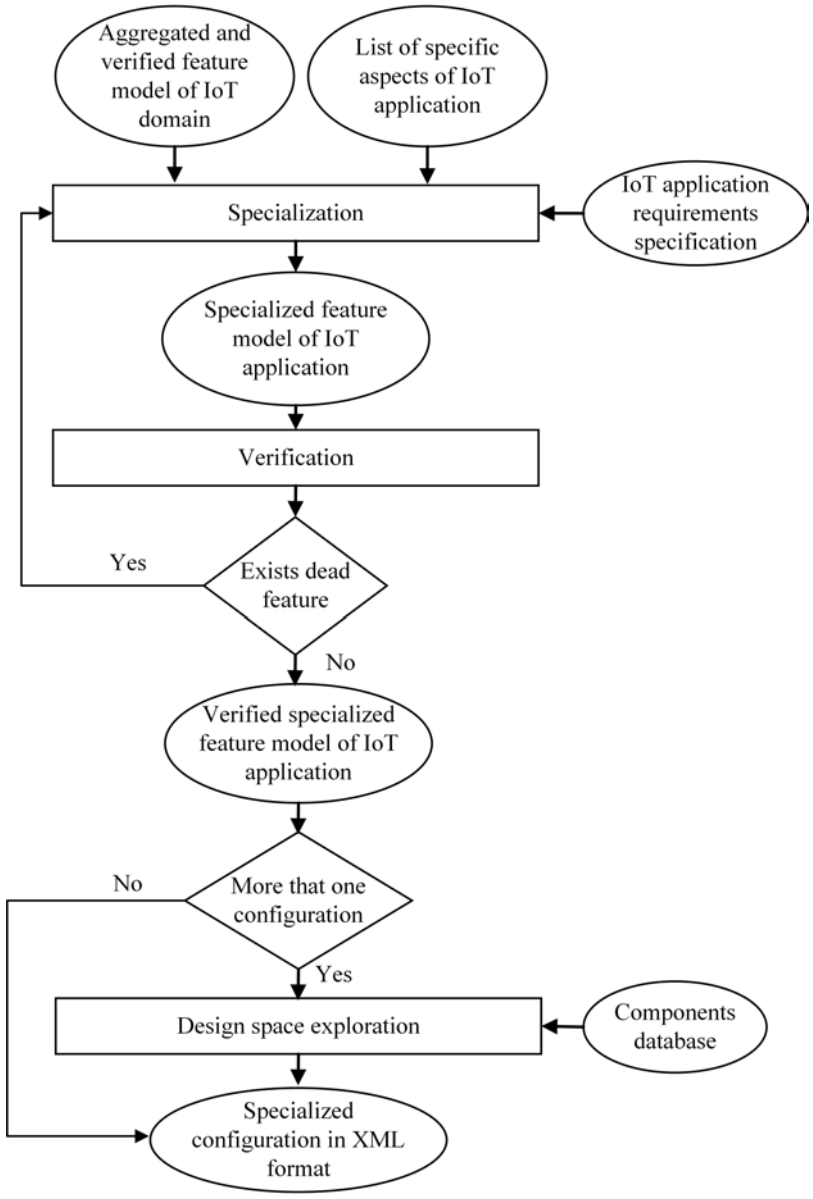


Figure 3.10. Specialization phase

Arbitrary edit. It describes merging operations that are not re-factoring, specialization or generalization.

Specialization is the merging mode of interest in the scope of this chapter. As it can be seen from the proposed specialization phase actions which are presented in Figure 3.10, the aggregated feature model is merged with specifications and a list of specific aspects of IoT-based application. Due to this, specialization Formula 5 must be rewritten as follows:

$$FM_{spec} \subset FM_{aggr} \otimes IoTAS \otimes LOSAoIoTA, \tag{7}$$

where FM_{spec} is the resulting specialized feature model, $IoTAS$ are the IoT application specifications, $LOSAoIoTA$ is a list of specific aspects of the IoT application.

Six categories of the specialization steps have been presented in [Czarnecki et al., 2005b]; these categories are for cardinality-based feature models. However, we believe that the three following specialization steps can be adapted for the Feature-RSEB feature models used in this work:

1. Removing a sub-feature from a group. Group cardinality and feature group of size k merges a set of k sub-features and indicates a choice of at least n_1 and at most n_2 distinct sub-features. A feature group can be changed by removing one of the sub-features, provided that $n_1 < k$. Sub-feature removing creates a new k-1 size group with cardinality

$$\langle n_1 - min(n_2, k-1) \rangle, \tag{8}$$

where $max(n, n')$ takes the minimum of the two natural numbers n and n'.

2. Selecting a sub-feature from the group. A specialization step can alter a feature group of size k with group cardinality $\langle n_1 - n_2 \rangle$ by selecting one of the sub-features, provided that $n_2 > 0$. And new k-1 size group will be created with cardinality:

$$\langle (max(0, n_1 - 1)) - (n_2 - 1) \rangle, \tag{9}$$

where $max(n, n')$ takes the maximum of the two whole numbers n and n'.

3. Assigning an attribute value. The specific value which is assigned to an uninitialized attribute, helps decrease the number of possible configurations. The value has to be the type of the attribute

The first and the second steps must be adapted and extended in order to use them for feature models which are used in this work. Adapted and extended steps of specialization are described below. The third step, assigning attributes and values, does not require an adaptation and extension, therefore, it can be used as presented above.

The specialization phase is shown in Figure 3.10. Two main steps of the specialization phase can be distinguished from Figure 3.10: aggregated feature model specialization and design space exploration in order to find the configuration which meets the IoT application's requirements best.

3.5.1. Creation of specialized model

At this step of the specialization phase, the aggregated feature model and the specifications and/or a list of specific aspects of the IoT application which must be developed are merged. This step is used to decrease the number of configurations (variability of PLs) presented by the aggregated model to the most relevant configurations for the IoT application which must be developed. Firstly, we must check if the aggregated feature model can represent all the specifications and/or aspects of the IoT application. If not, the aggregated feature model must be extended with features which could cover the missing specifications and/or aspects in order to present all the requirements of the IoT application which must be developed. The *Node* feature of the aggregated feature could be a good example of a feature which should be extended in order to present all the specifications of the IoT application.

Table 3. Rules to increase the number of configurations which are presented by the aggregated feature model [Alves et al., 2006]

Rule	Name
1	Convert Mandatory feature to Optional
2	Convert Alternative feature to Optional
3	Expand OR group by adding new member
4	Expand Alternative (XOR) group by adding new member
5	Create new OR group
6	Create new Alternative (XOR) group

For example, we have an aggregated feature model which is presented in Figure 3.9, and IoT application specifications say that nodes N1 and N2 must be used for the development of the IoT application. Moreover, specifications say that nodes can use P1, P2 and Pm communication protocols for data trans-

ferring. As it can be seen from Figure 3.9, in the aggregated feature model node N1 is presented as if it only can use P1 communication protocol while N2 uses P2. From this fact, we can see that the aggregated feature model cannot present all the specifications and/or aspects of the IoT application. The model cannot cover specifications where node N1 can use P2 and Pm communication protocols, and N2 deals with P1 and Pm. Therefore, *Node* feature of the aggregated feature model, which is presented in Figure 3.9, must be extended with the features which cover the missing specifications. After the extension, *Node* feature of the aggregated model should look as shown in Figure 3.11 in order to cover all the specifications of the IoT application which must be developed. The extended aggregated feature model is presented in Figure 3.12. As it can be seen from Figure 3.11, now, all the specifications and/or aspects of the IoT application are covered. By using the given examples of extensions, any feature of the aggregated feature model can be extended. Feature model extension by adding new features leads us to the fact that the extended aggregated feature model presents more configurations. Therefore, this step of specialization could be understood as a generalization where Formula 6 must be rewritten as follows and FM is understood according to Definition 1 (see Appendix A):

$$(\langle FD_{aggr}, CL_{aggr}\rangle \otimes IoTARS \otimes LOSAoIoTA) \subset \langle FD_{aggr\_ext}, CL_{aggr}\rangle, \quad (10)$$

where, FD_{aggr} is the aggregated feature model, FD_{aggr_ext} is the extended aggregated feature model, CL_{aggr} is a list of constraints of an aggregated feature model, $IoTARS$ are specifications of IoT application, $LOSAoIoTA$ is a list of specific aspects of IoT application.

As it can be seen from Formula 10, after the extension process, we get a new aggregated feature model $\langle FD_{aggr\_ext}, CL_{aggr}\rangle$, whose feature model structure has changed but the list of constraints remains the same. This is due the fact that at this step of the specialization phase, we check whether all the specifications and/or aspects of the IoT-based application could be presented by the aggregated model. Thus a list of constraints will be re-viewed at the next step of the specialization phase.

In Table 3, we present the rules adopted from [Alves et al., 2006] which can be used for the aggregated feature model extension in order to cover all the specifications and/or aspects of the IoT application which must be developed. The presented rules are used only if there is need to increase the number of configurations which are presented by the aggregated feature model and help implement the requirements of Formula 10.

At this step of specialization, it does not matter whether the extension has been carried out or not, we have a feature model in any case (see Figure 3.12) which presents all the possible configurations for the implementation of the specific IoT application whose domain was analysed in the domain model

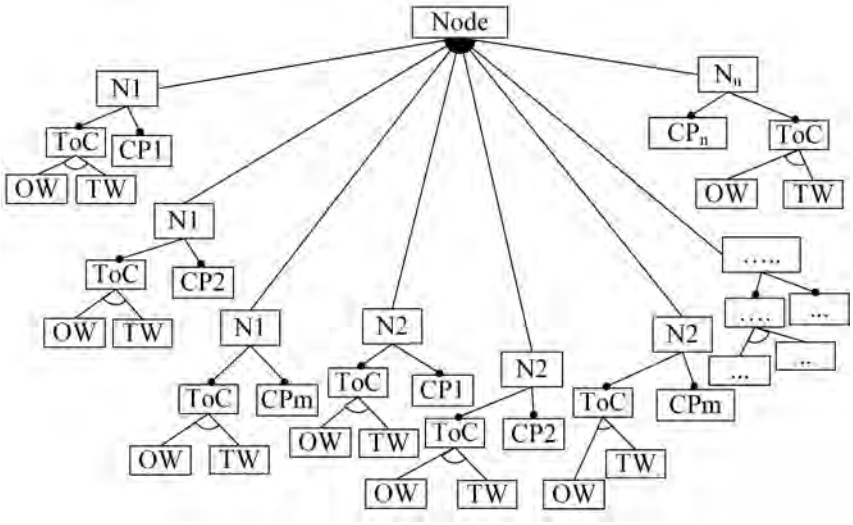


Figure 3.11. Extended *Node* feature of the aggregated feature model

development phase. The configurations which are presented by this model are not necessarily correct. Incorrect or illegal (not all the features presented by the aggregated feature model can be used for application implementation) configurations appear from the fact that during the aggregation and/or aggregated feature model extension process, no features and no constraints were deleted. Unnecessary and illegal configurations will be removed at the next step of the specialization phase.

After the merging process of the aggregated feature model and specifications and/or aspects of the IoT application, we obtain a feature model whose basic structure is shown in Figure 3.12. This model presents all the relevant configurations (all PLs which meet the specifications of the IoT application) to implement the IoT application whose specifications are given above.

As it can be seen from Formula 7, the specialization of the aggregated feature model according to the specifications and/or specific aspects of the IoT application is a step which decreases the number of configurations which are presented by the aggregated model. This is done by eliminating incorrect configurations by deleting the unused features and changing the variability of feature groups (see Figures 3.12 and 3.13). Furthermore, at this step, all the constraints of the aggregated feature model are reviewed and the unnecessary ones are deleted while new ones are added. New constraints come from the specifications and/or aspects of the IoT application where specifications and/or aspects of the IoT application could present which nodes must be used in order to implement the application and which communication protocol must be used for which node by describing what data transfer rate each node requires.

At this step of the specialization phase, the feature model is understood

according to Definition 1 (see Appendix A), specialization Formula 7 must be rewritten as follows:

$$\langle FD_{spec}, CL_{spec} \rangle \subset (\langle FD_{aggr}, CL_{aggr} \rangle \otimes IoTARS \otimes LOSAoIoTA), \quad (11)$$

where FM_{spec} is the resulting specialized feature model, CL_{spec} is a list of the resulting model constraints, FD_{aggr} is the aggregated feature model, CL_{aggr} is a list of aggregated feature model constraints. At this point, the aggregated feature model is a feature model which was created at the aggregation phase (see Figure 3.9) or during a previous step of the specialization phase (see Figure 3.12).

In order to achieve that, Formula 11 of the feature model specialization would be appropriate, and the following rules must be satisfied:

- The sum of FM_{aggr} aggregated feature model features must be greater than the sum of features of the FM_{spec} specialized model:

$$\sum FM_{aggr}(FeatNo) > \sum FM_{spec}(FeatNo), \quad (12)$$

  where *FeatNo* is the number of features.

- The FM_{spec} specialized feature model must have as many as possible constraints which are between the non-mandatory features (see Definition5 in Appendix A).

In order to achieve the requirements of Formula 12, we must adapt the 1^{st} and the 2^{nd} steps of the specialization which are presented by Czarnecki et al. [Czarnecki et al., 2005b] and described above. First of all, we suggested that feature variant points (see Definition 7 in Appendix A) of the merging aggregated feature model must be divided into two groups: XOR and OR. Each group represents a different variability. The number of configurations presented by the XOR group is equal to the feature number in the group (see Formula 13) while in the OR group, the number of configurations is calculated by Formula 14.

$$\sum (FeatureNumberInGruop) \quad (13)$$

and

$$\sum_{n=1}^{r} C_r^n = \frac{r!}{n!(r-n)!}, \quad (14)$$

were r is the number of group elements and $r \geq 2$. Because of 13 and 14, different rules can be applied to the different variant points of the feature model in the specialization process.

In order to decrease the variability presented by the XOR variant point, we suggest using three rules:

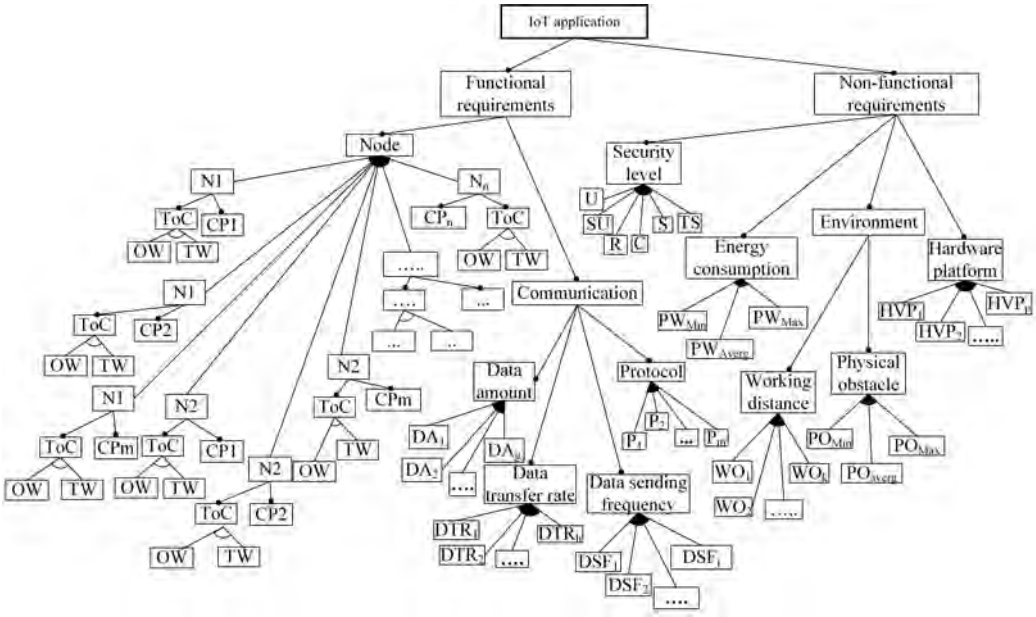


Figure 3.12. Extended aggregated feature model

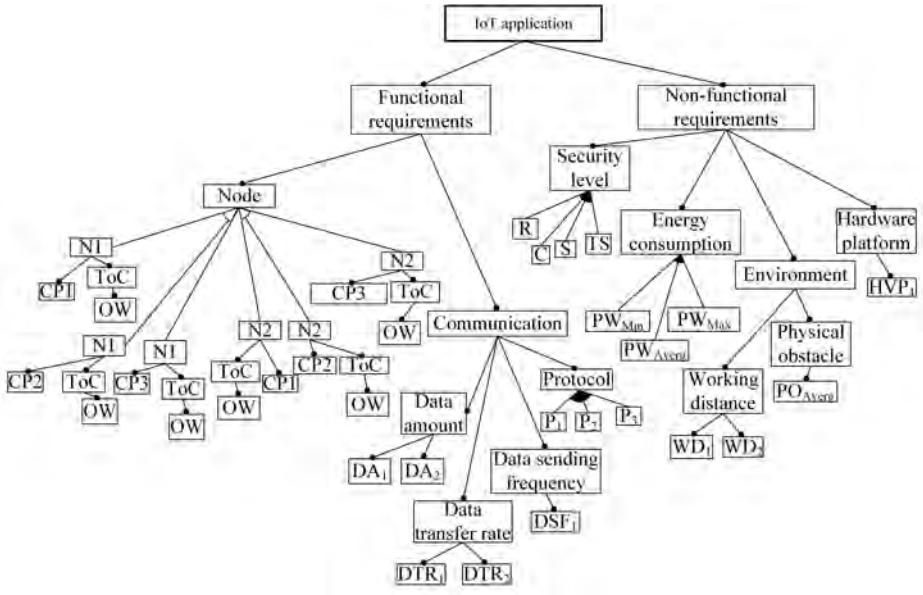


Figure 3.13. Specialized feature model for IoT application

- Feature of variant point can only be converted to a mandatory feature. In this case, feature conversion creates a new variant point with a new variability which is counted according to Formula 15. As can be seen Formula 8, which is used to present the variability of the cardinality-based feature models has been adapted to feature-RSEB feature models and rewritten as follows:

$$\sum(FeatNoInVP) - 1, \tag{15}$$

  were *FeatNoInVP* is the feature number of the variant point and FeatNoInVP > 1.

- A group feature can be deleted from the feature model.

- All *XOR* variant points can be deleted from the aggregated feature model.

We suggest four specialization rules which can be applied for the OR variant point:

- All variant point types can be changed to the XOR type. After that, the group variability is counted according to Formula 15.

- A feature of OR variant point can be converted to a mandatory feature. After converting one feature to mandatory, Formula 9, which describes the cardinality based feature model variant point is rewritten as follows:

$$2^{n-1} - 1, \tag{16}$$

  where n is the feature number of the variant point.

- Features of the OR variant point can be removed from the feature diagram. The variability of the new group, when one feature is removed, is counted according to Formula 13.

- All *OR* variant points can be deleted from the aggregated feature model

When specialization rules are applied in order to achieve the specifications and/or aspects of the IoT application, we get a specialized feature model (see Definition 22 in Appendix A) of the specific IoT application which presents all the relevant configurations for the implementation of this application; the presented configurations meet the specifications of the application fully. To illustrate the purpose, in Figure 3.13, we present a specialized feature model of the IoT-based application. The model has been created from the extended aggregated feature model presented in Figure 3.12. As it can be seen from Figure 3.13, the specialized feature model has a lot fewer features. Also, it can be seen, that some variant point has lost so many features that they are not

Table 4. Example of possible constraints between the features of a specialized feature model which can appear during the specialization process

No.	Feature	Constraint	Feature	No.	Feature	Constraint	Feature
1	P_1	Requires	PW_{Min}	5	P_1	Excludes	S
2	CP_2	Requires	P_2	6	P_1	Excludes	TS
3	CP_3	Requires	P_3	7	TS	Requires	P_2
4	N1	Requires	DTR_1	8	N2	Requires	WD_1
...	...	...	...	...	...	...	...

presented in the specialized model, whereas others have changed their types. Also, the newly created specialized model has a new list of constraints. This list is adapted to cover all the specifications and/or aspects of the IoT application which must be developed. Several possible constraints of a specialized feature model are presented in Table 4. After specialization, the created specialized feature model must be verified to ensure that it is correct.

If all the presented steps were applied correctly and the model verification was successful, a correct specialized feature model is created. Figure 3.13 presents an example of the specialized feature model which was created from the aggregated model (see Figure 3.12). The specialized feature model presents only configurations which meet the requirements of the IoT application which must be developed. As it can be seen from Figures 3.12 and 3.13, the specialized model has much fewer features than the aggregated model. Also, the specialization process reduces the configurations number from the millions presented by the aggregated feature model to several hundred or even fewer as presented by the specialized feature model. The number of configurations depends on several aspects: the feature model being created in the domain models development phase, the IoT application domain, the IoT application specifications, and on how accurately proposed steps of specialization were applied.

From the configurations which are present in the specialized feature model, we have to choose one configuration which matches the specifications and requirements of the IoT application best because different performances of the IoT application can be ensured by each configuration. To do that, we suggest using the design space exploration process where the configuration is selected by adding additional aspects according to the IoT application's requirements. For example, these aspects could be as follows: the configuration must use the lowest possible amount of energy and ensure the highest possible security level. Moreover, we understand these aspects as IoT application requirements for QoS, which must be ensured. The design space exploration is the last step

of the specialization phase, after which we get a specialized configuration, from which, the code framework of the IoT application, which must be developed, will be generated.

3.5.2. Design space exploration

After the specialization step, we have obtained a specialized feature model of the specific IoT application; this specialized feature model usually describes several possible configurations allowing to implement the IoT application. These configurations are understood as a design space of the IoT application. All these configurations consist of two parts: a) one which stays the same for all configurations; b) one which varies for each configuration. The part which is the same for all configurations is described by mandatory features (see Definition 5 in Appendix A). The part which is different for each configuration is described by features which are in variant points (see Definition 7 in Appendix A). As it can be seen from the created specialized feature model (see Figure 3.13), the parts which are different for each configuration describe IoT application nodes, communication protocols, and security levels. Due to this fact, at the design space exploration (DSE) step, the specialized feature model is understood as a list of n different configurations whose generic structure is shown in Figure 3.14.

$$FM_{spec} = \left\{ \begin{array}{l} \{\ Conf_1\ \} \\ \{\ Conf_2\ \} \\ \{\ .....\ \} \\ \{\ Conf_n\ \} \end{array} \right\}$$

Figure 3.14. A list of configurations created from a specialized feature model

As it can be seen from Figure 3.13, the *Node* feature presents only the types of nodes which can be used for the development of an IoT application. Furthermore, m action modules can be used to implement each node (see Explanation 2, e.g. N1 node can be implemented by using AM_1, AM_2 and other action modules). In addition, each action module performs the same actions but has different specifications. The same is with communication modules because a specialized feature model presents only communication protocols which can be used by a node. For example, there can be k options to choose a communication module for a N1 node where k can be different for each communication protocol presented in the specialized feature model. The number of action and communication modules which can be used for one node implementation comes from the *components database* (see Chapter 3.5.3). Components database is used to hold information about the action (sensors, actuators, etc.) and com-

$$Conf_1 = \left\{ \begin{array}{l} \{\ N1\_1, CP1\_1, N2\_1, CP2\_1, RoC1\ \} \\ \{\ N1\_1, CP1\_1, N2\_1, CP2\_2, RoC1\ \} \\ \{\ N1\_2, CP1\_2, N2\_1, CP2\_1, RoC1\ \} \\ \{\ N1\_2, CP1\_2, N2\_1, CP2\_2, RoC1\ \} \\ \{\ N1\_1, CP1\_2, N2\_1, CP2\_1, RoC1\ \} \\ \{\ N1\_1, CP1\_2, N2\_1, CP2\_2, RoC1\ \} \\ \{\ N1\_2, CP1\_1, N2\_1, CP2\_1, RoC1\ \} \\ \{\ N1\_2, CP1\_1, N2\_1, CP2\_2, RoC1\ \} \end{array} \right\}$$

Figure 3.15. Extended $Conf_1$ configuration of a specialized feature model. RoC1 represents the part of configuration which remains unchanged.

munication modules which can be used in the development of IoT applications. The number of action and communication modules depends on the hardware platform which is chosen to be used for the IoT application implementation, e.g. in a specialized feature model, it is *HVP1*.

Due to these facts, each configuration presented by the specialized feature model must be extended in such a way that the node type (e.g. N1 and N2 features from the specialized feature model, Figure 3.13) must be changed to the specific action module names which come from the components database. The same is with features CP1, CP2, and CP3 which describe communication protocols.

For example, we have a $Conf_1$ configuration (from the specialized feature model presented in Figure 3.13) which requires that for the implementation of the IoT application, we must use N1 node which uses CP1 communication protocol, and N2 uses CP2 communication protocol. In addition, we have a components database which describes that for the implementation of N1 node, we can use N1_1 and N1_2 action modules, for N2 we can use N2_1, and for the implementation of communication protocols CP1_1 and C1_2 we use communication modules for CP1, and CP2_1 and CP2_2 for CP2. In Figure 3.15, we present a *Conf1* configuration after an extension, which was extended according to the presented information. At this point, each module from the components database is taken with all of its characteristics the way these components are presented in Figure 3.19, thus it should be understandable as a list and presented e.g. as $CP1\_1[char_1, char_2, ..., char_n]$. Due to the readability, we present these components as it is shown in Figure 3.16. The selected components from the components database with all their characteristics are necessary because these characteristics will be used for the selection of the best configuration for the given application.

As can be seen from Figure 3.15, the parts of configurations which must be changed describe nodes and communication protocols. Also, from Figure 3.15, we can see that a newly extended configuration presents all the possi-

$$
FM_{spec} = \left\{ \begin{array}{l}
Conf_1 = \left\{ \begin{array}{l}
\{\ N1\_1, CP1\_1, N2\_1, CP2\_1, RoC1\ \} \\
\{\ N1\_1, CP1\_1, N2\_1, CP2\_2, RoC1\ \} \\
\{\ N1\_2, CP1\_2, N2\_1, CP2\_1, RoC1\ \} \\
\{\ N1\_2, CP1\_2, N2\_1, CP2\_2, RoC1\ \} \\
\{\ N1\_1, CP1\_2, N2\_1, CP2\_1, RoC1\ \} \\
\{\ N1\_1, CP1\_2, N2\_1, CP2\_2, RoC1\ \} \\
\{\ N1\_2, CP1\_1, N2\_1, CP2\_1, RoC1\ \} \\
\{\ N1\_2, CP1\_1, N2\_1, CP2\_2, RoC1\ \}
\end{array} \right\} \\
Conf_2 = \left\{ \begin{array}{l}
\{\ N1\_1, CP1\_1, N2\_1, CP2\_1, RoC2\ \} \\
\{\ N1\_1, CP1\_1, N2\_1, CP2\_2, RoC2\ \} \\
\{\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \}
\end{array} \right\} \\
Conf_n = \left\{ \begin{array}{l}
\{\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \} \\
\{\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \}
\end{array} \right\}
\end{array} \right\}
$$

Figure 3.16. Specialized feature model after configurations extension. RoC presents the rest of the configuration

ble variants to implement nodes with different communication protocols. The number of variants depends on the action module number which can be used to implement each node, and on the communication module number which can be used to implement node communication protocols in a given configuration. For example, we have a configuration which presents two N1 and N2 nodes, where N1 uses CP1 communication protocol and N2 uses CP2. Moreover, in order to implement N1 node, we can use a different action modules from the components database, b is used for N2, and c communication modules are used for CP1 communication protocol, and d is used for CP2. In this way the number of different variants presented by one configuration after extension would be counted as $a{\cdot}b{\cdot}c{\cdot}d$. After the extension of every configuration which is presented by the specialized feature model, we get a specialized feature model which is derived from a list of configuration lists whose structure is presented in Figure 3.16.

Now, each newly created configuration, which is presented in Figure 3.16, presents a configuration with real hardware components which can be used for the development of the IoT application whose domain has been analysed and whose functional and non-functional requirements feature models were created. As it was mentioned before, hardware components, which describe the same actions have different characteristics. Moreover, e.g. from Figure 3.16 we can see that $Conf_1$ and $Conf_2$ configurations describe the same action and communication modules, but the remaining part of the configurations is different. In this different part of the configurations, the security level is presented which must

$$
FM_{spec} = \left\{ \begin{array}{l}
Conf_1 = \left\{ \begin{array}{l}
\{\ N1\_1, CP1\_1, N2\_1, CP2\_1, RoC1\ \}, ES11 \\
\{\ N1\_1, CP1\_1, N2\_1, CP2\_2, RoC1\ \}, ES12 \\
\\
\{\ N1\_2, CP1\_2, N2\_1, CP2\_1, RoC1\ \}, ES13 \\
\{\ N1\_2, CP1\_2, N2\_1, CP2\_2, RoC1\ \}, ES14 \\
\{\ N1\_1, CP1\_2, N2\_1, CP2\_1, RoC1\ \}, ES15 \\
\{\ N1\_1, CP1\_2, N2\_1, CP2\_2, RoC1\ \}, ES16 \\
\{\ N1\_2, CP1\_1, N2\_1, CP2\_1, RoC1\ \}, ES17 \\
\{\ N1\_2, CP1\_1, N2\_1, CP2\_2, RoC1\ \}, ES18
\end{array} \right\} \\
Conf_2 = \left\{ \begin{array}{l}
\{\ N1\_1, CP1\_1, N2\_1, CP1\_1, RoC2\ \}, ES21 \\
\{\ N1\_1, CP1\_1, N2\_1, CP1\_2, RoC2\ \}, ES22 \\
\{\ \ldots\ldots\ldots\ldots\ldots\ldots\ \}, \ldots
\end{array} \right\} \\
Conf_n = \left\{ \begin{array}{l}
\{\ \ldots\ldots\ldots\ldots\ldots\ldots\ \}, ESn1 \\
\{\ \ldots\ldots\ldots\ldots\ldots\ldots\ \}, ESnm
\end{array} \right\}
\end{array} \right\}
$$

Figure 3.17. A configurations list with estimates. ES presents the calculated estimate of configuration

be ensured by communication modules. Due to these facts, those hardware components which can be used for the IoT application implementation have different characteristics and each configuration describes the different working mode of these components, and each newly created configuration can ensure a different level of performance of the IoT application. This performance of the IoT application can be understood as the QoS of IoT application.

Due to this reason we must select one configuration from the specialized configurations which are presented in Figure 3.16. This configuration should describe the performance (ensure QoS) of the IoT application in the best possible way according to the specific criteria which are important for this application. Therefore, each configuration must be evaluated according to QoS criteria of IoT application which must be developed. Moreover, the criteria usually contradict each other, e.g. the increase in the security level versus the reduction of the power consumption.

After the evaluation of configurations, we get a list of configurations (see Figure 3.17) where every configuration is denoted by its own estimate which was calculated according to QoS criteria. This estimate shows how well the configuration satisfies a specific IoT application's QoS criteria. Now, when we have a list of configurations with estimates, one configuration with a minimum or a maximum estimate must be chosen which depends on the priorities of the IoT system. The selected configuration describes the hardware components and the working modes of these components which should be used for the

development of the IoT application and can ensure the best QoS according to the given criteria. In Figure 3.17, we present how a list of configurations should look after evaluation.

The evaluation problem of the configurations which are presented in Figure 3.16 can be understood as multi-criteria optimization in the design space of the IoT application. Optimization helps evaluate configurations according to the assigned QoS criterion, which comes from the IoT application requirements for the specific performance. A basic multi-criteria optimization problem can be formulated as follows [Zionts, 1988]:

$$\underset{X=(x_1,...,x_n)\in D}{minF(X)} = [f_1(X), f_2(X), ..., f_m(X)]^T, \tag{17}$$

where D is a bounded domain in the n-dimensional space R^n, $\mathrm{X} = (x_1, x_2, ..., x_n)$ is a vector of variables, the functions $f_j(X) : R^n \to R^1$ are criteria and m is the number of criteria.

In order to solve the problem formulated according to Formula 17, multi-criteria decision analysis algorithms are used. For configuration evaluation, we suggest using the Pareto optimality [Karimpour and Ruhe, 2017], which gives engineering the optimal solution rather than the mathematical one. When solving the Pareto optimality, we search for the Pareto optimum point of the Pareto set (see Explanation 3) [Marler and Arora, 2004]. In our case, the Pareto set is a list of configurations which is presented in Figure 3.17 and can be used for the IoT application implementation. In order to find the Pareto optimum, first of all, configurations must be evaluated. We assume that the weighted sum method [Gass and Saaty, 1955] would be a good choice to estimate configurations. The notion of the weighted sum method is:

$$\underset{X\in D}{min} \sum_{i=1}^{m} w_i F_j(X) \tag{18}$$

where w_i is the weighting coefficient of i criteria, $0 < w_i \leqslant 1$, $\sum\limits_{i=1}^{m} w_i = 1$, m is the number of criteria, $F_j(X)$ is the j-th configuration's criteria and j is a list of criteria according which optimization must be performed. The coefficient of a criterion would be assigned by the application developers according to the QoS requirements.

Explanation 3 *A point X^* is said to be a strict Pareto optimum or a strict efficient solution for the multi-criteria problem if and only if no $X \in D$ such that $F_i(X) \leqslant F_i(X^*)$ for all $i \in \{1, 2, ..., n\}$, with at least one strict inequality.*

In order to get estimates of different configurations according to QoS criterions, various dimensional units must be evaluated and added which are described by objective functions. Thus, usually, unit normalization is used. We

suggest that normalization should be performed by Formulas 19 and 20 when minimizing or maximizing the objectives [Jakob and Blume, 2014]:

$$F_i^{norm}(X) = \frac{max(F_i(X)) - F_i(X)}{max(F_i(X)) - min(F_i(X))} \text{ for objectives to be minimised} \quad (19)$$

and

$$F_i^{norm}(X) = 1 - \frac{max(F_i(X)) - F_i(X)}{max(F_i(X)) - min(F_i(X))} \text{ for objectives to be maximazed.} \quad (20)$$

In order to evaluate configurations, Formula 18 must be rewritten as follows:

$$\min_{X \in D} \sum_{i=1}^{m} w_i F_j^{norm}(X) \quad (21)$$

In the proposed optimization process of the DSE step of the specialization phase, F(X) used in Formula 21 is one of the configurations from the list which is presented in Figure 3.16. Optimization objectives would come from the components database, and optimization criteria weights would be assigned by the system developers according to the IoT application requirements for performance.

In order to create a list of configurations which is presented in Figure 3.14, a specialized feature model must be parsed. Then, the specialized feature model is parsed, and we have all the configurations presented by this model separately. The next step is to create a list of configurations presented in Figure 3.16 which presents configurations with real components which can be used for the IoT application development. In order to do that, the components from the components database must be selected. When selecting components from the components database, first of all, from each configuration, the following information must be extracted (as an example, we used a specialized feature model presented in Figure 3.10): action modules types (N1 and N2), communication protocol of the action module (CP1, CP2 and etc.), hardware platform (HVP1), security level for each communication protocol (R, C, S and TS) and physical obstacle (PO_{Averg}). For each configuration, this information is extracted separately and can be different.

Parsed information is used to select hardware components from components database where components from the components database would be selected by comparing the extracted information and the information which is contained in the components database. In order to select an action module from the components database, two rules must be implemented: 1) the extracted type of the action module must coincide with the name of the action in the components database; 2) the extracted hardware platform's name

must coincide with the hardware platform's name which is in the action module description in the components database. The rules of components database creation are presented in Chapter 3.5.3.

The action module selection from the components database can be presented as follows: (for example, we used a specialized feature model presented in Figure 3.10 and a components database presented in Figure 3.19): $\underbrace{N1}, \underbrace{HVP1} = \underbrace{N1}\_1[\underbrace{HVP1}, ...]$, where marked information matches in accordance with action module selection rules, and $[HVP1, ...]$ presents action module characteristics which are in the components database. Also, it can be seen that in order to select an action module from the components database, the marked parts must coincide, and this is due to the database filling rules (see Chapter 3.5.3). When an action module is extracted from the components database, the next step is to extract communication modules which can be used to implement the communication protocol which is associated with this action module. For example, in one of the configurations which is presented by a specialized feature model, *N1* action module is associated with *CP1* communication protocol.

In order to select a communication module from the components database, the following rules are used: 1) the extracted communication protocol must coincide with the name of the communication module in the components database; 2) the extracted hardware platform's name must coincide with the hardware platform's name which is in the action module description in the components database; 3) the extracted security level must coincide with the security level which is used in the communication module description; 4) the extracted physical obstacle must coincide with the physical obstacle which is used in the communication module description.

The communication module selection from the components database can be presented as follows (for example, we used the specialized feature model(see Figure 3.10), a components database presented (see Figure 3.19) and S security level): $\underbrace{CP1}, \underbrace{HVP1}, \underbrace{S}, \underbrace{PO_{Averg}} = \underbrace{CP1}\_1[\underbrace{HVP1}, \underbrace{S}, ..., \underbrace{PO_{Averg}}]$ where the marked information matches as the communication module selection rules require. $[HVP1, S, ..., PO_{Averg}]$ presents communication module information characteristics, which is in the components database. For each action module, all the possible communication modules are extracted from the components database. After this has been done, we have obtained a list which presents an action module and communication modules associated with it. The created list could look as follows: $N1[char_{N1}][CP1\_1[char_{CP1\_1}], CP1_2[char_{CP1\_1}], ...]$, where*char* represents the characteristics of the selected module.

After having completed all the selections, we have a list whose structure is presented in Figure 3.16. When such a list has been created, the next step is the evaluation of each configuration in order to create the list presented in Figure 3.17. As it was mentioned before, the evaluation of configurations

```
<IoT_application>
 <Hardware_platform> VHP1</Hardware_platform>
 <N1>
      <Action_module>N1_1</Action_module>
           <Communication module_name = "CP1_1" >
           <Security_level>SecL_1</Security_level>
           <TypeOfCommunication>OW</TypeOfCommunication>
           </Communication>

 </N1>
 <N2>
      <Action_module>N2_1</Action_module>
           <Communication module_name = "CP2_1" >
           <Security_level>SecL_2</Security_level>
           <TypeOfCommunication>OW</TypeOfCommunication>
           </Communication>

 </N2>
</IoT_application>
```

Figure 3.18. Specialized configuration in the XML format

will be performed by using Formula 21, where the necessary information for optimization is bound to come from components characteristics lists which were selected in the previous step. For example, if one of the optimization criteria were *energy*, then all the information related to energy would be used for the optimization, and this information would be multiplied by the weight of these criteria.

When we have obtained a list presenting configurations and their estimates which were calculated according to QoS requirements of the IoT application, by using the Pareto optimality, we need to choose one configuration (specialized configuration) from this list. However, there may be a situation that none of the evaluated configurations satisfies the QoS requirements; therefore, the weights of the optimization criteria must be changed, and the optimization process must be repeated (see Figure 3.13). When the optimization process has been completed, the one configuration which meets the QoS requirements of the IoT application the best is selected. This configuration contains information about hardware components and their working mode which are important for code generation. Due to this fact, from the selected configuration, the following information must be parsed: the hardware platform's name, the action module's names, the communication modules' names, and the security level which must

be ensured by communication modules (for each communication module, it can be different) and the working modes of communication which must be ensured by communication modules. This information is parsed directly into an XML format file as for the specialized configuration, such a file structure is presented in Figure 3.18. In Figure 3.18, we show what one of the specialized configurations for the specific application after the optimization process may look like; the configurations were obtained from the specialized model (see Figure 3.13). From the specialized configuration, we can see that *N1* and *N2* nodes are described with real hardware components which must be used for the implementation of these nodes and for the working modes of these hardware components. The specialized configuration presented in Figure 3.18 is the output of the specialization step which will be used for the code framework generation of the IoT application.

The generated XML file of the specialized configuration consists of the following elements:

IoT_application is the IoT application's name whose specialized configuration is created. This specialized configuration attribute depends on what the root feature of the aggregated feature models is called (see Figures 3.9).

Hardware_platform presents the hardware platform which should be used for the development of the IoT application. It is VHP1 in Figure 3.18.

N1 specifies the IoT node which is presented in the specialized model (see Figure 3.13) and whose main components are presented below. The number of the nodes depends on specification of IoT applications. According to specialised model it is two: N1 and N2.

Action_module describes the name of the concrete action module which must be used for the development of the node of the IoT application and was selected during the design space exploration process. The name of the action module comes from the components database, and in the presented specialized configuration N1_1 corresponds to N1 node and N2_1 corresponds to N2 in Figure 3.18.

Communication specifies attributes which should be used to ensure the performance of the communication and data transfer of the action module (e.g Action_module in Figure 3.18).

module_name describes the concrete communication module (e.g. CP1_1 for N1_1 action module in Figure 3.18) which was selected from the components database during the design space exploration process.

Security_level presents the security level (the working mode) which is assigned during the design space exploration process and must be ensured by the communication module (e.g. SecL_1 for CP1_1 in Figure 3.18).

TypeOfCommunication describes what type of communication is used by action module. This specialized configuration attribute comes from the specialized model (e.g. OW for N1_1 action module in Figure 3.18).

3.5.3. Components database

The components database is used in the IoT application's design spare exploration process and contains the necessary information for it. This database contains a list of hardware modules which can be used to implement the IoT applications. These modules can be sensors, actuators, communication modules and other devices which are used or can be used in IoT applications. Every hardware module included in the components database must be described with a list of characteristics which are ensured by the module. A list of characteristics varies depending on the type of module, e.g. a sensor has different characteristics from those of an actuator or a communication module. Some characteristics come from the description of modules while others are calculated by experiments. Most characteristic of modules can be used as an optimization criterion. The components database can be extended at any time by adding new hardware modules.

In addition to the specific characteristics, every module must have a characteristic parameter which shows for which hardware platform this module is designed (e.g. Arduino, Gadgeteer, etc.).

For the names of hardware components which are inserted into the components database, we suggest using the combination from the type of module and the real name of the module. For example, we have a Wi-Fi communication module whose name is RS21, the name of this module in the components database must be Wi-Fi_RS11; for the temperature sensor whose name is SHT1x it would be Temperature_SHT1x. Such naming helps identify and search for components as it was shown in the design space exploration process step of the specialization phase. Moreover, from Figure 3.18, it can be seen that the proposed module naming is used to describe the specialized configuration which will be used in the IoT application framework generation phase (see Chapter 3.6).

We suggest that the modules which are included in the components database must have the following characteristics in addition to the abovementioned hardware platform, and this suggested list is inconclusive:

- Sensor: accuracy (AC)– how accurately it can perform measurements; working range (WR) -what is the measurement diapason; power consump-

```
CP1_1{HVP, DTR_m, PC, SecL_1 , WD, EpIU , Env}
CP1_2{HVP, DTR_m, PC, SecL_2 , WD, EpIU , Env}
CP_n{HVP, DTR_m, PC, SecL_2 , WD, EpIU , Env}
. . . . . .
N1_1{HVP, AC, WR, PC}
N1_2{HVP, AC, WR, PC}
N_m{HVP, AC, WR, PC}
. . . . . .
AC_1{HVP, PC, SR}
AC_k{HVP, PC, SR}
. . . . . .
Other IoT modules
```

Figure 3.19. Fragment of the components database

tion (PC) – what is the average power consumption while performing the measurement.

- Actuator: power consumption (PC) is the average power consumption which is used to perform an action; sample rate (SR) denotes how fast an action can be performed.
- Communication module: data transfer rate (DRT_m) is the maximum data transfer which can be ensured with this module; power consumption (PC) is the average energy power consumption during data transferring; working distance (WD) is the maximum distance across which data can be transferred; security level ($SecL$) is used to present near which working modes of communication modules some of characteristics were calculated.

In addition, we suggest that the communication module must have additional characteristics which are important for the process of design space exploration and may have a major impact on configuration evaluation results. These additional characteristics would be calculated experimentally. These characteristics will be used to show the relationships related to the security level and energy consumption, energy consumption and the environment, energy consumption and the security level as well as the environment. After the experiments, we should get the following characteristics: power consumption per information unit ($EpIU$), which was transferred to another device in different environments by using different security levels. Also, environment characteristic Env must be added too in order to show in which environment $EpIU$ characteristics were calculated. To describe environment characteristic, we suggest using one of three units: min, average, max as it is described in Chapter 3.3 and presented in Figure 3.6a.

In Figure 3.19, we present a fragment of a components database where CP is a communication module, N is a sensor, AC is an actuator, and the other abbreviations are as above-mentioned.

3.6. Generation phase

After the specialization phase, a specialized configuration of the IoT application which must be developed was created. A specialized configuration is as if an XML document which contains all-important aspects which are used for the implementation of the IoT application. In addition, specialized configuration contains all the information which is needed to generate the code framework of the IoT application.

The proposed code generation has a simple structure and it can be understood as a text processor aimed to generate a code framework. The programming language and the architecture design of the generated framework is determined by the code template libraries (see Chapter 3.6.1). If the framework must be generated in the C language, the generated framework will be implemented in C as well. This generation method can be easily extended by adding new code fragments to the code template library or a new code template library in the code template repository (see Chapter 3.6.1).

Figure 3.20 shows the process of code generation; as it can be seen, the proposed code generator takes on a specialized configuration (see Figure 3.18) as the input. As it was mentioned before, this configuration contains all the necessary information for code framework generation of the IoT application. The first step which is done by the code generator is the parsing of the specialized configuration. In Figure 3.20 it can be seen that the specialized configuration is parsed into two lists: the action modules and the hardware platform. The action modules list contains all the information which is related to the action modules: the action module's name, the communication module's name, the communication module's working mode and the type of communication. This information will be used for code selection from the code template library at the next step of code generation. The hardware platform is selected as a separate variable due to the fact that, before code generation, the code template library must be selected from which the code will be generated in the next steps of the code generation process.

When the hardware platform information is parsed, the next step, which is performed by the code generator, is the code template library selection from the code template repository. The hardware platform's name is used for the code template library selection the where hardware's and code template's names must match. The selected code template library contains methods related to a specific hardware platform. The methods describe the performance of action modules, the functionality of communication modules and data transferring. Apart from the methods in the code library, also information about the

programming language of the hardware platform is presented: depending on whether the programming language is object-oriented or not, the extension of the output file is contained (see Figure 3.23). These two parameters will be used in the final stage of the code framework generation (see Figure 3.20).

The next step, after the code template library selection, is the method selection from the code template library for actions and communication modules, and data transferring. These methods describe the performance and the communication type functionality of modules. The selection is done through the name of the module and the name of code template parts (the tag name) which describes all the methods which are used to describe this module's performance. The code generator is the search code description part where the description parameter's name in the code template library is the same as the module's name (see Chapter 3.6.1). When such a parameter has been found, all the methods which are presented in this part are copied from the code template library. The same is done with the methods which describe the communication module's functionality.

In order to select methods that describe the type of communication functionalities (one or two-way), and which are associated with the communication module, two parameters in the code template library must match: the communication protocol for which these methods can be used and the communication type. Due to this fact, in order to present methods which describe the type of communication, in the code template libraries, a different notation is used (see Chapter 3.6.1). Because of this specificity, in order to select methods which describe the type of communication for a specific protocol, the code generator must determine what communication protocol is used by the selected communication module. This is done by creating the name of communication name where e.g. CP1_1 communication module name consists of the communication protocol name *CP1* and 1 real name of the communication module (see Chapter 3.6.1). When the communication protocol is known, the methods which describe the type of communication functionality can be selected from the code template library because other necessary information comes from the node parameter list which has been created at the first step of code generation (see Figure 3.20). Moreover, the same methods which describe the type of communication functionality can be used by several different communication protocols.

After the code selection has been done, we get a list of methods where the methods describe action and communication module functionality and the data transferring functionality (the type of communication). Such a list would be created for each node which is presented in the specialized configuration. In Figure 3.21, we show an example how such lists could look for nodes which are presented in the specialized configuration (see Figure 3.18). As it can be seen from Figure 3.21, the code generator selects not only the methods but also

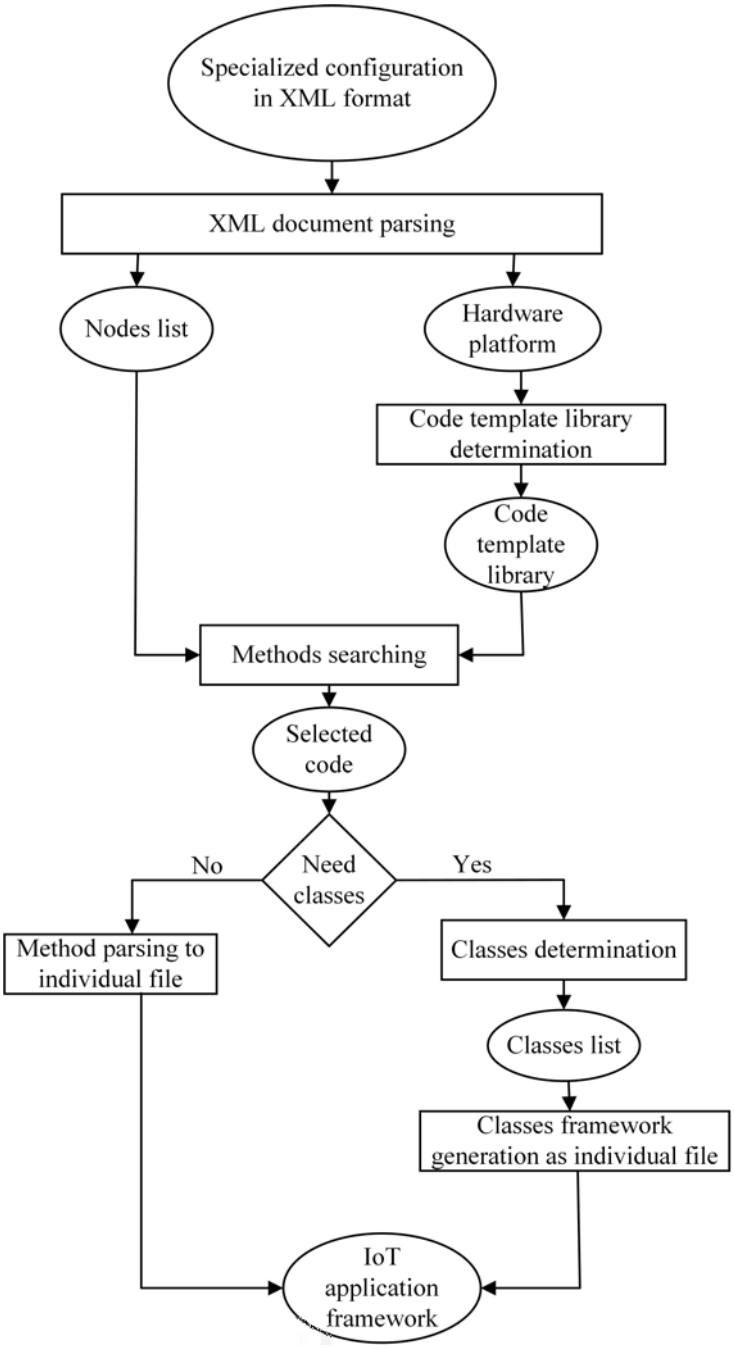


Figure 3.20. Code generation process

the information about the programming language which is used in the selected hardware platform. In Figure 3.21, *Class* presents if the programming language

```
Class = "Yes"
Extension = ".ext"
N1:
method1_N1_1
method2_N1_1
method1_CP1_1
method1_CP1_1(SecL_1)
method1_OW
method2_OW

N2:
method1_N2_1
method2_N2_1
method1_CP2_1
method1_CP2_1(SecL_2)
method1_OW
method2_OW
```

Figure 3.21. Selected code components from the code template library

is object oriented or not, *Extension* shows the extension which must be used for the output files of code generation.

The last step of code generation is performed after all the information for the IoT application framework generation has been selected. First of all, from the selected information, the structure of the output code is determined. This means that the type of the programming language must be determined, whether it is object oriented or not. As it can be seen in Figure 3.20, on this information, the operations of the code generation depend. If the programming language is not object-oriented, then the selected code which is shown in Figure 3.20 would be divided into two individual files with *.ext* extension. The code would be divided according to the node name, and the output file would be named according to the node name (N1 or N2 as it is presented in the specialized configuration). If the programming language is object-oriented, as shown in Figure 3.21, then this code is divided into classes, where the class name is the node name, as shown in Figure 3.22. Each class will be exported as an individual file where the output file name will be the same as the class name. The generated code files are treated as the code framework which is used for the IoT-based application implementation.

The output of the generation phase is the code framework for the IoT application implementation. The generated framework contains methods which are not interconnected; thus the framework is not a fully functional application. A fully functional application will be implemented in the implementation phase

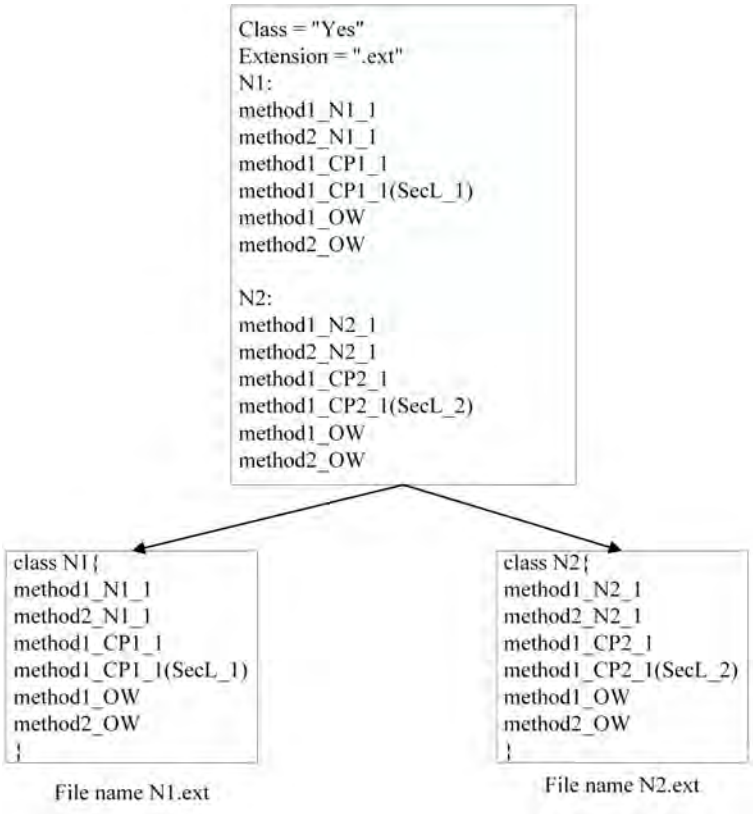


Figure 3.22. Generated framework for IoT application.

(see Chapter 3.7). All the necessary logical links and extends will be added which are needed according to the requirements of the application in order to get a fully functional IoT application.

3.6.1. Code template repository

The code template repository contains the code template libraries where every code template library is as an individual file for a different hardware platform where each code template library is named according to a hardware platform's name. Each library contains the code methods which are used to describe the performance of the specific hardware components.

The code template library is created by application engineers and can be reused for different implementation of the IoT application where the hardware platform is used whose methods this particular library contains. Each library should contain as many methods as possible for each specific hardware unit. In Figure 3.23, we present the structure of each code template library. Such a structure allows extending the code template library with new methods at

any time. All the methods which are put into the code template library are fully implemented according to the requirements of the programming language of the hardware platform.

The code template library is an XML document whose structure is shown in Figure 3.23. The four parts of the code template library from the suggested structure can be distinguished: meta-data, action_module, communication_module and communication_type. The first part describes the meta-data information about the programming language of the hardware platform. As it can be seen from Figure 3.23, this information shows what type of programming language is used in the selected hardware platform, and what file extension is used by hardware platform programming language for the output files. This meta-data information is used to determine the structure of generated framework.

The second part of the code template library contains methods and variables which describe the performance of action modules. The methods which describe one action module's performance would be grouped by using an XML tag where the tag would be named according to the module name; this tag name in Chapter 3.6 is treated as a description parameter. From Figure 3.23, it can be seen that the methods which describe CP1_1 action module's performance are grouped by using a tag with CP1_1 name. We suggest that, for tag naming, one should use action module names exactly the way they are presented in the components database (see Chapter 3.5.3).

The third part of the code template library contains the methods and variables which describe the communication module's operation. The grouping of methods of the communication modules is the same as that of the action modules. The methods are grouped by the communication module's name whose functionality they describe, and where name is the same as in the component's database. Most methods which are used to describe the functionality of communication modules have a parameter which allows to assign the security level in which the modules must communicate. We suggest that this parameter would be marked as *Sec_L*, and in the implementation phase this parameter would be a change in the security level which is presented in the specialized configuration by the application developer. For example, for CP1_1 communication module which is presented in the specialized configuration (see Figure 3.18), such a parameter would be *SecL_1*. However, not all communication modules allow such methods which can control the security level of the communication module because these modules must be preconfigured by using specific software. Thus, the application developers must configure such communication modules to use security level which is presented in specialized configuration.

We suggest that the fourth part of the code template library should contain methods and variables which describe data transferring. Our analysis showed that two different communication modules which operate in the same commu-

```
<Platform name="HVP">
<Meta>
<Output file="ext"/>
<Class type = "yes"/>
</Meta>
<Code>
  <N1_1>
      method1_N1_1
      method2_N1_1
  </N1_1>
  <N2_1 >
      method1_N2_1
      method2_N2_1
  </N2_1>
</Code>
<Communication_module>
  <CP1_1>
      method1_CP1_1
      method1_CP1_1(Sec_L)
  </CP1_1>
  <CP2_1 >
      method1_CP2_1
      method1_CP2_1(Sec_L)
  </CP2_1>
</Communication_module>
<Communication_type>
  <OW protocols="CP1, CP3" >
      method1_OW
      method2_OW
  </OW >
  <OW protocols="CP2">
      method1_OW
      method2_OW
  </OW >
</Communication_type>
</Platform>
```

Figure 3.23. Fragment of a code template library

nication protocol use the same methods for data transmission. In addition, our analysis showed that different communication protocols could use the same methods for the same type of data transferring but the methods which describe

one-way and two-way data transferring are different for the same protocol. Due to these facts, for the description of data transferring methods, the extended XML tag notation must be used where the XML tag would be concluded from the tag name and attribute and should look as follows: <OW protocols="">. The tag name should represent for which data transferring type the grouped methods are used, and the protocol attribute should present for which protocols these methods can be used. We suggest that for such a tag name we should be used abbreviations: OW for one-way and TW for two-way. Moreover, not all the methods and attributes are known when the methods which describe data transferring are added to the code template library as these methods contain such parameters as IP addresses, the socket port and other which are changing depending on the network configuration of the IoT application. Thus, we suggest, that these parameters would be marked as *???* (e.g. SERVER_IP = ???) so that they would be easier to find for the system developer(s) and change them at the implementation steps of the generated framework.

The suggested code template library consists from the following elements:

Platform specifies an attribute which is used to present a hardware platform whose code components are saved in the code library

name presents the name of the hardware platform whose code components are saved in code library, e.g. as HVP in Figure 3.23.

Meta specifies the code template part which is used to present information about the programming language.

Output specifies an attribute which is used to describe an extension of the output file of the programming language.

file is an attribute which describes the programming language output file(s) extension whose code components are saved in the code library, e.g. as ext in Figure 3.23.

Class specifies an attribute which is used to demonstrate if the programming language requires classes.

type is an attribute which describes if the programming language whose code components are stored in code library requires classes or not. This attribute can have two values: yes – the programming language requires classes; no – the programming language does not require classes.

Code specifies the code template part which is used to store the programming methods of sensors, actuators, and other hardware. N1_1 specifies the name of the action module whose programming methods are included in the code library. The action module names are the same as they are

described in they components database. In Figure 3.23 are presented to action modules: N1_1 and N2_1.

Communication_module specifies the code template part which is used to present the communication module's programming methods of the hardware platform.

CP1_1 specifies the name of a communication module whose programming methods are included in the code library. Communication module names are the same as they are described in the components database. In Figure 3.23 are presented to action modules: CP1_1 and CP2_1.

Communication_type specifies the code template part which is used to store the programming methods which are used for the data transfer of various communication protocols.

OW specifies what type of communication is used to present the programming methods. We distinguished between two types of communication: OW – one-way and TW – two-way.

protocols are attributes showing for which communication protocol(s) data transfer methods are used. As it was mentioned earlier, the same data transfer methods can be used for several different communication protocols.

3.7. Implementation phase

At this phase of the proposed IoT applications development methods, the working applications are created by extending the generated IoT application framework with the necessary functionally which was not generated in the generation phase of the proposed IoT application implementation method. All the actions which are done at the implementation phase are based on widely used application project management steps. We think that the closing process [Li and Tang, 2014] of application management processes describes the actions of the implementation step best. In order to describe the actions which are performed in the implementation phase, we divide them into three steps: framework testing, application implementation, and application testing.

Framework testing. In the proposed implementation phase, the framework-testing step is used to check if all the components of the IoT application are generated in the generation phase. If the components are missing, they should be implemented in the next step of the implementation phase. In other words, the system developer checks that the code generated at the generation phase is correct for the specialized model (see Figure 3.13) and the specialized configuration(see Figure 3.18). Thus this step of the implementation phase can be understood as the generated code verification which is done by the system developer.

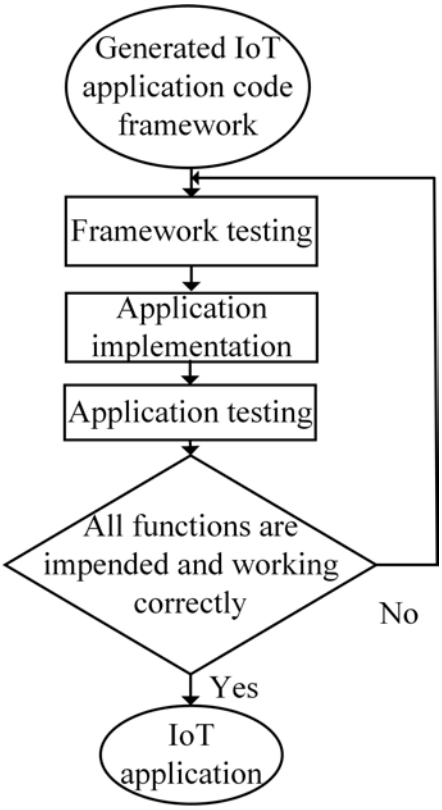


Figure 3.24. Implementation phase

Application implementation. This step of the implementation phase is used to create a fully functional IoT application. In order to carry out this action, a logical connection must be added between the methods of generated code framework of IoT application. Furthermore, this step is responsible for the extension of the generated framework. We understand extension as an implementation of IoT application component(s) which have been determined during the framework-testing step.

Application testing. Testing activities are aimed at discovering errors in the applications after they have been created so that the action can be taken to remove them. Further, testing also checks if all the functionalities of the IoT-based application are implemented and operating as they should. There are many testing techniques, but at this point we do not suggest any particular techniques, and we think that the testing technique must be chosen by the application tester according to which parts of the application s/he wants to test.

The implementation phase is the last phase of the proposed IoT application implementation method (Figure 3.2), after which we get a fully working

IoT application.

3.8. Conclusions

1. The IoT application implementation method based on feature modelling has been presented. The presented method consists of five phases which cover the development of the application from modelling to code application implementation.

2. The method models the possible variability (PLs) of the application implementation in its domain. In order to do that, the basic structure of the functional and non-functional requirements feature models has been proposed which is used in the analysis of the application domain and covers the following aspects of IoT application development: security, energy consumption and the heterogeneity of devices and communication protocols. The variability is presented in the aggregated feature model of the IoT application domain.

3. When using the presented specialization steps, an aggregate feature model can be adapted to the specifications of the specific IoT application. To achieve that, the aggregated feature model and specifications of the IoT application are merged. The output of the merging process is the specialized feature model of the specific IoT application.

4. A methodology to analyse the configurations presented by the specialized feature model according to the IoT application requirements for QoS with respect to heterogeneity, security and energy consumption has been presented. The proposed evaluation methodology enables to suggest communication protocol(s), specific hardware and the working mode of communication module(s) which should be used for IoT application implementation.

5. The code framework generation process from the specialized configuration based on the code template library has been presented.

4. DEVELOPMENT OF IoT-BASED HEALTHCARE APPLIACTION

This chapter is dedicated to the development of an IoT-based application by using the proposed IoT application development method. For the purpose of proof-of-concept, we use an IoT-based healthcare-oriented application. During the case study, all the phases of the proposed IoT application development methods were implemented.

4.1. Software tools

This section describes the software tools used for modelling, verification and the experimental set-up of the feature models.

The S.P.L.O.T. tool (Software product lines on-line tools) [Mendonca et al., 2009] allows creating feature models. Feature models can be created by using Feature-RSEB notation (see Appendix B). This tool has additional functionalities which provide information about the created feature model, such as if the model does not have any dead feature (see Definition 14 in Appendix A), how many configurations (product line) are presented by this model as well as other important information. The created feature model is presented in the textual format, which makes it harder to read when the model has a huge number of features. By using S.P.L.O.T. software, the created feature model can be exported as an SXFM file which can be imported in other software. S.P.L.O.T. was used for the verification of feature models.

The FAMILIAR (for FeAture Model script Language for manipulation and Automatic Reasoning) is a language for importing, exporting, composing, decomposing, editing, configuring, reverse engineering, testing, and reasoning about (multiple) feature models. The creators of this language also proposed a software tool, which supports this language and ensures all the features of this language. The tool allows to create feature models in the graphical mode which is more user-friendly if compared with S.P.L.O.T. because it is better readable when the feature model has a huge number of features. The graphical notation of this tool is based on Feature-RSEB notation. The FAMILIAR tool also has an option which provides information about the feature model, just like S.P.L.O.T., but the amount of information is significantly smaller. However, this tool can import and export the created feature models in different formats, including an SXFM which is used by S.P.L.O.T. FAMILIAR software and has been used for the modelling of feature models.

4.2. Case study: IoT-based healthcare application

This case study presents our proposed IoT application development method (see Figure 3.2) for the developing of the IoT-based healthcare application. The proposed method was used to implement an application, which

performs measurements of vital information about humans and sends this information for further processing. In order to ensure heterogeneity, different communication protocols and sensors were used. This chapter is based on [Jusas et al., 2016, Venckauskas et al., 2016a, Venckauskas et al., 2016b, Venckauskas et al., 2016c]

4.2.1. Introduction

According to [Hussain et al., 2015, Kamel Boulos and Al-Shorbaji, 2014, Mukherjee et al., 2014], IoT-based healthcare application can be presented as three-level architecture. Figure 4.1 presents the architecture of IoT-based healthcare application. Each architecture layer of the application describes different levels of the application development. The healthcare-oriented application module containing adequate facilities (such as the patients' health histories database, the patients' database and the application server) represents the highest level. The standard Internet module with connecting facilities represents the intermediate level. The distributed sensors network represents the lowest level; in fact, it represents a sub-net of a distributed sensors network along with additional facilities. In terms of the application, we treated it as a body area network (BAN) [Filipe et al., 2015]. Our proposed IoT application implementation method is oriented towards the opportunity to present the variability of configurations which can be used for the creation of the BAN layer of healthcare application; thus the BAN layer is the layer of interest in this case study. This layer is responsible for providing the initial data.

In the healthcare applications, there is a need to collect a huge number of data from different parts of the patient's body [Memon et al., 2014]. The collected data from different sensors is to be combined and presented as a common structure before being transferred to the healthcare-oriented application module. As the modules BAN and healthcare-oriented application typically are located in the remote places, the sub-module heterogeneous network concentrator (see Figure 4.1) also ensures the interfacing facilities with the standard Internet module.

In addition, measured data protection during the transfer is also an important aspect of the IoT-based healthcare application. Typically, the measured data is private and not to be allowed for capturing and tampering. Commonly, for data protection, communication protocols and their security mechanisms are responsible; however, the security level may be different for various applications or for the specific cases of the same application, e.g. for different BAN. Therefore, the healthcare applications are concerned with choosing the adequate protocol from the set of available ones. As it is highlighted in the structure of BAN, the different sensors may require the different protocols. Typically, multi-functional sensors require more advanced protocols, because they generate more data, which requires a bigger data transfer rate in compar-

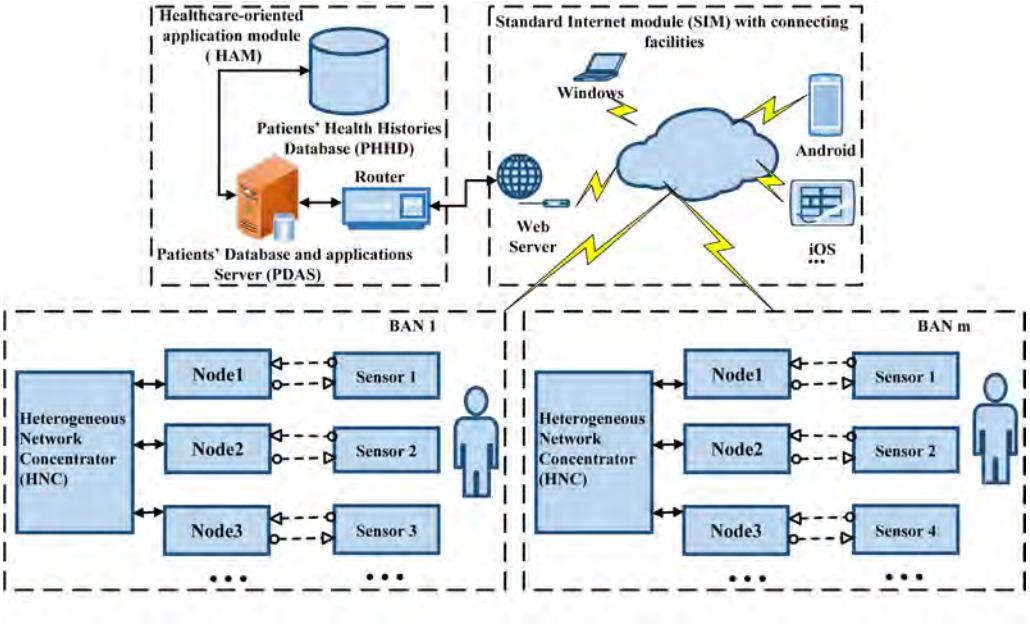


Figure 4.1. Three-level architecture of the IoT-based healthcare application

ison to simple sensors. Therefore, the sensor's type predefines the use of the adequate protocol.

The procedure of obtaining the initial data and then transferring it according to the selected protocol is also concerned with energy consumption. The latter highly depends on the mode of using BAN (intensiveness of the data stream, sensor type, protocol to be used, the state of the environment, such as the level of noise, etc.). For telemedicine applications, energy issues are important too, because there are many battery-charged medical devices in use. All these factors in building the healthcare application should be taken into account at the initial phase when the requirements for the application are to be stated. Therefore, the requirements fall into two categories: non-functional (related to the energy and security, environmental factor) and functional (related to the measured data and its transfer). We should note that the medical interpretation and the use of the data is not the concern of this case study.

Now, we have a description of the reference architecture of the IoT-based healthcare application. This information is sufficient to start describing the implementation of our IoT application implementation method.

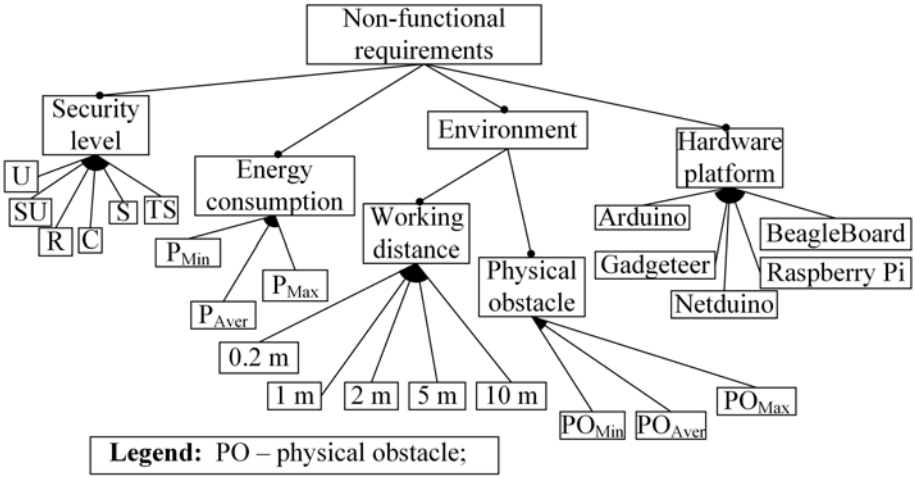
4.2.2. Modelling of IoT-based healthcare BAN layer domain

As it was mentioned in Chapter 3.3, this phase of the proposed IoT application implementation method is responsible for the analysis and modelling of

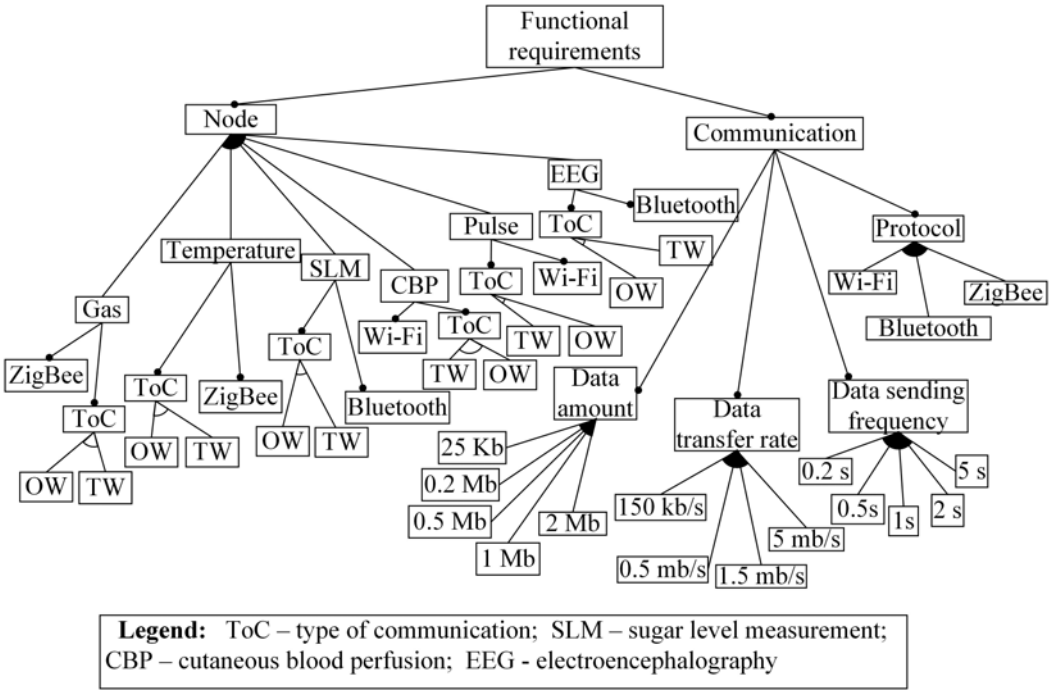
the IoT application domain. In our case, this domain is an IoT-based healthcare BAN layer. For modelling purposes, we will be using the proposed functional and non-functional requirements feature models (see Chapter 3.3 and Figure 3.6).

As it was mentioned above, our proposed method is oriented towards a BAN layer of the IoT-based healthcare application. As can be seen from Figure 4.1, two main components can be distinguished which must be analysed in this domain and which are important for the application. The first component is sensors, actuators and other action modules which are or can be used in this domain and are necessary for the implementation of the IoT-based healthcare application. Secondly, these are the communication protocols which are or can be used to transfer data from action modules to the heterogeneous network concentrator. These two components of the body area network application were components of interest during the processes of analysis of IoT-based healthcare BAN domain. Moreover, domain analysis was oriented towards information which could help feature models adhere to the proposed functional and non-functional requirements.

According to the proposed non-functional requirements feature model, information about the hardware platforms and the working distance of the action modules is the most important in order to complete this model. Domain analysis was oriented towards analysis of this information as other features of the model remain the same as proposed in Chapter 3.3. The domain analysis shows that there are many hardware platforms which can be used for the development of the IoT-based healthcare BAN layer. Therefore, in order to reduce the number of possible hardware platforms for the application development, we chose platforms according to several criteria: the hardware platform must be open source and must present an opportunity to connect possible sensors and actuators which can be used for application development. By using these criteria, five most popular hardware platforms were chosen (see Figure 4.2a). Moreover, the domain analysis shows that the body area network action modules are usually concentrated in a small area. Action modules can be located directly on a patient or on his clothes, from where the measured data is sent to e.g. a smartphone; also, this area can be all the house of the patient. Due to this fact, the working distance of action modules, we have chosen from 0.2m to 10m as it is shown in Figure 4.2a. In Figure 4.2a, we present the feature model that specifies a set of non-functional requirements of IoT-based healthcare-used BAN layer's domain. In Table 6, we present the constraints relationships between the features of non-functional requirements feature model. At the domain analysis stage, constraints between features come from the analysis itself or expert knowledge. Constraints among features of non-function requirements feature model, which are presented in case study, are based on [Venckauskas et al., 2014a, Venckauskas et al., 2014b]. Table 5 shows the cre-



(a) Non-functional requirements feature model of IoT-based healthcare application



(b) Functional requirements feature model of IoT-based healthcare application

Figure 4.2. The created functional and non-functional requirements feature models of IoT-based healthcare applicationn

ated model's characteristics where we can see that the created feature model covers 562,185 non-functional configurations (PLs), which can be used for the development of IoT-based healthcare BAN layer.

Table 5. Functional, non-functional and aggregated feature model characteristics. Models statistics obtained using the S.P.L.O.T. tool. * - Variability Degree is the number of valid configurations divided by 2^n,where n is the number of features in the model

No.	Data	Non-functional model	Functional model	Aggregated model
		Statistics		
1	Features	29	54	83
1.1	-Optional	0	0	0
1.2	-Mandatory	6	18	26
1.3	- Grouped	22	35	57
1.4	- Groups	5	11	16
2	Tree Depth	4	6	7
3	Extra constraints	9	10	27
4	Distinct extra constraints variables	12	12	24
5	Clause Density	0.8	0.8	1.7
6	CNF Clauses	49	99	173
		Results		
1	Consistency	Consistent	Consistent	Consistent
2	Dead Features	0	0	0
3	Common Features	7	7	19
		Metrics		
1	Count Configurations	562185	1568352	4.34E7
2	Variability Degree* (%)	1.0472E-1	8.7061E-9	1.4352E-14

The creation of the functional requirements feature model of the IoT-based health-care BAN layer domain requires more extensive analysis. All the lowest layer features must be established while in the non-functional model case some of the features remain the same as they are presented in Figure 3.6a. Healthcare BAN layer domain analysis has shown that action modules for the communication between each other and heterogeneous concentrators, most of the time, use the wireless communication protocols. After conducting analysis of the wireless communication protocols which can be or are used in IoT-based healthcare BAN layer, we have chosen only three most popular standard protocols nowadays. More theoretically possible wireless communication protocols are presented in Table 1. The data sending frequency feature shows how often

Table 6. Constraints relationships of the functional and non-functional requirements feature models of the IoT-based healthcare BAN layer domain

Functional model constraints				Non-functional model constraints			
No.	Feature	Constraint	Feature	No.	Feature	Constraint	Feature
1	Temperature	Requires	ZigBee	1	U	Requires	P_{Min}
2	Gas	Requires	ZigBee	2	SU	Requires	P_{Min}
3	SLM	Requires	Bluetooth	3	R	Requires	P_{Aver}
4	EEG	Requires	Bluetooth	4	C	Requires	P_{Aver}
5	Pulse	Requires	Wi-Fi	5	S	Requires	P_{Max}
6	CBP	Requires	Wi-Fi	6	TS	Requires	P_{Max}
7	Bluetooth	Excludes	5 Mb/s	7	PO_{Min}	Requires	P_{Min}
8	ZigBee	Excludes	2 Mb/s	8	PO_{Aver}	Requires	P_{Aver}
9	ZigBee	Excludes	1 Mb/s	9	PO_{Max}	Requires	P_{Max}
10	ZigBee	Excludes	0.5 Mb/s				

the measured data must be transferred, and this feature depends on the application requirements. Thus, we have chosen five times in seconds, which, we think, is most relevant for the healthcare BAN layer (see Figure 4.2b). The data amount feature shows how much data is generated by the sensor or actuator performing one action. This is closely related to action modules' actions which are performed by them, moreover, this depends on the type of action modules as it was mentioned before. In order to cover this feature, we have chosen several possible data amounts from very small to very big, and the possibilities are shown in Figure 4.2b. The data transfer rate depends on the data amount and data sending frequency features; thus, we have chosen such data transfer rates which cover all the possible configurations of the data amount and the data sending frequency features.

The domain analysis of the IoT-based healthcare BAN layer shows that there are many action modules which can be or are actually used for the development of such kind of applications. These action modules can measure various parameters and perform various actions. From these numerous possibilities, we have chosen six most popular ones, and these sensors and actuators cover all the possible types of functionalities which were mentioned above. The communication protocol and type of communication must be assigned to every sensor and actuator. Because, at this stage of application, we do not know this information, and in order to fill the communication feature we choose the most frequently occurring communication protocols for such types of sensors and actuators whereas the type of the communication feature is left exactly as it is. In Figure 4.2b, we present a feature model of the functional requirements of the IoT-based healthcare BAN layer. The constraints of the functional feature model are based on Table 1 and [Venckauskas et al., 2014b]. In Table 5 we show

the characteristics of the created functional requirements model, where we can see that the created feature model covers 1,568,352 functional configurations (PLs) which can be used for the development of the IoT-based healthcare BAN layer.

From the characteristics of created functional and non-functional requirements feature models (Chapter 3.3 and Figure 3.6), we can see that by using the proposed feature models with minor extensions, especially it can be seen from the non-functional requirements feature model, the IoT-based application domain can be analysed fully. Both models describe many possible variants (the variability of PLs) of functional and non-functional requirements which can be used for the IoT-based healthcare BAN layer implementation. From these different configurations, we think, it is possible to choose at least one configuration which fully describes the IoT-based healthcare BAN layer in functional and non-functional aspects.

The created non-functional and functional requirements models (see Figure 4.2) present the IoT-based healthcare BAN layer domain from functional and non-functional perspectives. In order to get all the possible configurations so that to implement the application in this domain, these two models must be combined. This is performed during the aggregation phase of the proposed IoT application development method. Also, the created feature models can be reused for the creation of other applications which are in the same or in similar domain to the case study by using models as they are presented in Figure 4.2 or employing minor changes.

4.2.3. Aggregated model of healthcare BAN domain

In Figure 4.3, we present the aggregated feature model which combines both functional and non-functional requirements of feature models. As it can be seen, this new feature model has a new root feature (see Definition 3 in Appendix A), which now is *"Healthcare"* according to the application type (see more in Chapter 3.4).

The aggregated feature model describes all the possible configurations (PLs) which can be used for the development of the IoT-based healthcare BAN layer according to the information which was collected during the analysis of the BAN domain. As can be seen from Figure 4.3, all the features presented by the created non-functional and functional requirements feature models are presented by the aggregated feature model. Aggregation of two models leads to the fact that new constraints can be distinguished between the features of the non-functional and functional requirements of feature models. The aggregated feature model contains all the former constraints between the non-functional model features and the functional requirements model features. The constraints between the features of the newly created feature model are shown in Table 7. As can be seen from Tables 7 and 6, the aggregated feature model contains

Table 7. Relationships between the features of the aggregated feature model

No.	Feature	Constraint	Feature	No.	Feature	Constraint	Feature
1	Temperature	Requires	ZigBee	15	U	Requires	P_{Min}
2	Gas	Requires	ZigBee	16	SU	Requires	P_{Min}
3	SLM	Requires	Bluetooth	17	R	Requires	P_{Aver}
4	EEG	Requires	Bluetooth	18	C	Requires	P_{Aver}
5	Bluetooth	Excludes	TS	19	S	Requires	P_{Max}
6	Bluetooth	Excludes	C	20	TS	Requires	P_{Max}
7	Bluetooth	Excludes	5 Mb/s	21	PO_{Min}	Requires	P_{Min}
8	ZigBee	Excludes	2 Mb/s	22	PO_{Aver}	Requires	P_{Aver}
9	ZigBee	Excludes	1 Mb/s	23	PO_{Max}	Requires	P_{Max}
10	ZigBee	Excludes	0.5 Mb/s	24	Wi-Fi	Requires	P_{Max}
11	Bluetooth	Requires	P_{Aver}	25	ZigBee	Requires	P_{Min}
12	ZigBee	Excludes	TS	26	Pulse	Requires	Wi-Fi
13	Bluetooth	Excludes	S	27	CBP	Requires	Wi-Fi
14	TS	Requires	Wi-Fi				

more constrains, which shows that the functional and non-functional features are closely related and required or excluded by each other.

The appearance of a new constraint requires that the aggregated feature model must be verified so that to ensure that it is correct. After the verification, we get an aggregated feature model which is correct (see Table 5) and presents the domain of the IoT-based healthcare BAN layer fully, according to the collected information during the model's development phase. For model verification, we have used the S.P.L.O.T. tool which automatically searches for the dead features and identifies them. The S.P.L.O.T. tool for the search of dead features uses the binary decision diagram (more see at [Mendonca et al., 2009]).

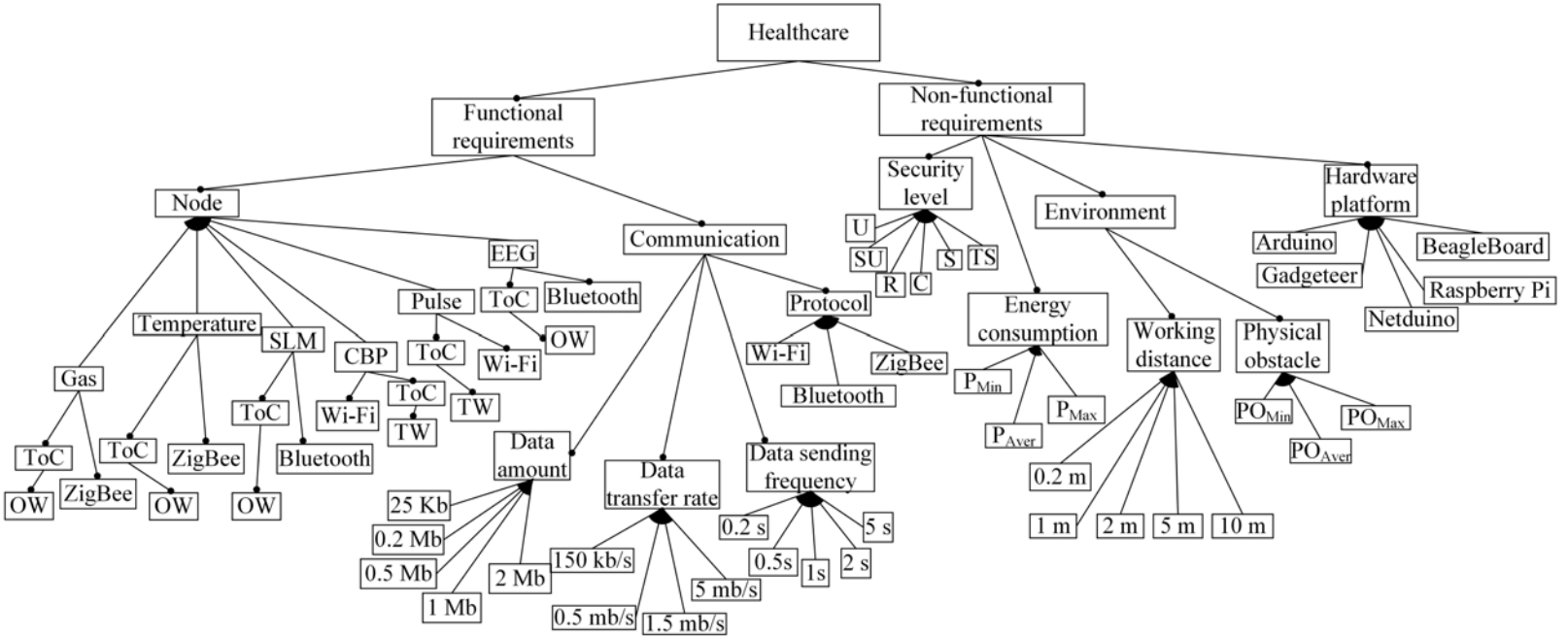


Figure 4.3. Aggregated and verified feature model of IoT-based healthcare application domain

In Table 5,we present the characteristics of the aggregated and verified feature model of the IoT-based healthcare application domain. From Table 5, we can see that by using the developed aggregated feature model, the great number of possible configurations which can be used to implement the IoT-based healthcare application are presented, and all these configurations combine functional and non-functional requirements. At this point, the aggregated feature model describes all the IoT-based healthcare BAN layer domain. Most configurations presented by this model are not relevant for the application which is bound to be created. Therefore, these irrelevant configurations must be removed, and, in order to do that, the aggregated feature model must be merged with the application specifications which must be implemented and whose code framework will be generated during the generation phase of the proposed IoT application development method. The merging of specifications of an application and the aggregated feature model creates a specialized feature model of the IoT-based healthcare application. Merging is performed during the specialization phase of the proposed IoT application development method. Moreover, the created aggregated feature model can be used for the development of other applications which are in the same domain or in similar domains. Due to the fact that this model presents a great number of configurations (PLs) to implement the BAN layer in its domain, the re-usability of the model is thus ensured.

4.2.4. Specialization of healthcare applicaiotn's BAN layer

The first step of the specialization phase of the proposed IoT-based application development method is the aggregated feature model merging with the IoT-based healthcare BAN layer's specifications. For example, the purpose case study application must ensure the following specifications:

1. Action modules: temperature, gas and pulse;
2. Hardware platform: Gadgeteer;
3. For the implementation of the application, Wi-Fi, Bluetooth and ZigBee communication protocols can be used;
4. The security level must be no lower than restricted (or R);
5. The action module's working distance: Pulse up to 10 meters, Gas up to 5 meters, Temperature up to 1 meter;
6. The data amount generated by the action modules: pulse up to 2 MB, gas up to 25 KB, temperature up to 1.5 MB;
7. The action module's data sending frequency: pulse 0.5 s, gas 2 s, temperature 5 s;

8. The action module's communication type: pulse: two way, gas and temperature: one way.

After outlining the BAN layer requirements which must be implemented, the next step is to check if the aggregated feature model can cover these specifications. According to the specifications, three types of sensors and actuators must be used for the development of the IoT healthcare application's BAN layer. These sensors and actuators do not have any assigned specific communication protocols, and, at this stage, each sensor and actuator can be implemented by using all the three communication protocols. As it can be seen from Figure 4.3, in the aggregated feature model, these components are presented as if they are only using one communication protocol. In other words, the aggregate feature model presents only one opportunity to choose communication protocols for the temperature sensor while, according to the application specification, there should be three of them. Therefore, the *Node* feature of the aggregated feature model should be extended in order to fulfil all the possible specifications of the IoT healthcare BAN layer. This fact leads to the extension of the aggregated feature model because during the modelling phase we only used the most common communication protocol for certain action module. In Figure 4.4, we present only the *Node* feature of the aggregated model after extension. Only three IoT-based healthcare application action modules are shown in Figure 4.4, because other sensors and actuators presented in the aggregated feature model are irrelevant according to the specifications of the case study application. After the extension, all the specifications of the IoT-based healthcare application can be covered by using other features of the aggregated feature model.

After the extension of the necessary features of the aggregated model of the IoT-based healthcare application's BAN layer, we can cover all the application specifications for the BAN layer. The next step of the specialization phase after the aggregated feature model extension is the specialization process of the extended aggregated feature model. Specialization is performed according to the rules presented in Chapter 3.5.1. After completing specialization, we get a specialized feature model which describes all the possible configurations (PLs) to implement the IoT-based healthcare application's BAN layer according to whose specifications it was created. In Figure 4.5, we present the created specialized feature model of the IoT-based healthcare BAN layer; this model presents all the possible correct configurations allowing to implement the BAN layer.

Relationships between the features of the specialized feature model are shown in Table 8. As it can be evidently seen, some of the constraints remain the same as in the aggregated feature model (see Table 7) while others come from the application specifications. Due to application's specifications, some features are not present in the specialized feature model because it was not necessary for application development. A good example of such feature is

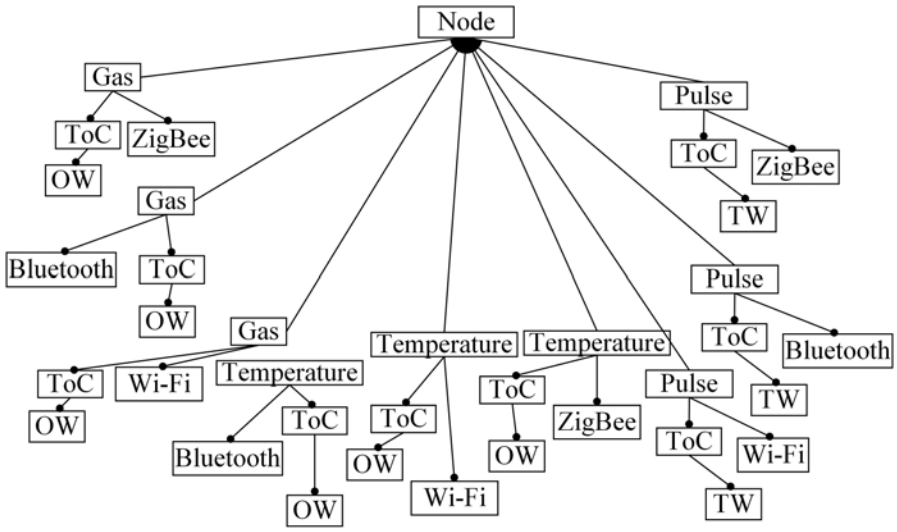


Figure 4.4. Extended *Node* feature of the aggregated feature model

Physical obstacle, in the aggregated model it has three features while in the specialized model it has only one feature. Similar aspects may be noted with *Node* feature, in Figure 4.4 it is shown that *Pulse* sensor can be implemented by using all the three communication protocols, while in the specialized model only Wi-Fi protocol was preserved. Such elimination was performed due to the application specifications for the pulse sensor and the limitation of the data transfer rate of other communication protocols. Afterwards, the specialization model was verified to ensure that the created model is correct.

The created specialized feature model of the IoT-based healthcare application presents all the possible configurations (the variability of possible PLs) which can be used for the development of the BAN layer of the application. In Table 9, we present characteristics of the specialized feature model. As it can be seen from Table 9, the specialized feature model presents 120 configurations which fully cover all the requirements of the IoT-based healthcare application. Thus we can see that the specialization dramatically decreases the number of configurations which can be used for an IoT-based healthcare application and which were presented by the aggregated feature model. Each configuration presented by a specialized feature model (see Figure 4.5) can be divided into two parts: the part which is the same for all the configurations and the part which changes. In our case, the part which changes describes the communication protocols which can be used by action modules, and the security levels which can be used by the communication protocol for data protection during the data transfer. For example, from the IoT-based healthcare application specifications and Figure 4.5, we can see that the *gas* sensor can be implemented by using three

Table 8. Relationships between features of the specialized feature model

No.	Feature	Constraint	Feature	No.	Feature	Constraint	Feature
1	Pulse	Requires	Wi-Fi	14	0.5 MB	Requires	1.5 Mb/s
2	Temperature	Requires	5 s	15	2 MB	Requires	5 Mb/s
3	Temperature	Requires	Wi-Fi	16	25 KB	Requires	150 Kb/s
4	Temperature	Requires	Bluetooth	17	Gas	Requires	25 KB
5	Temperature	Requires	1 m	18	ZigBee	Excludes	TS
6	Temperature	Requires	0.5 MB	19	Pulse	Requires	5 MB/s
7	Bluetooth	Excludes	C	20	Pulse	Requires	0.5 s
8	Bluetooth	Excludes	S	21	Gas	Requires	150 Kb/s
9	Bluetooth	Excludes	TS	22	Gas	Requires	Wi-Fi
10	Temperature	Requires	1.5 Mb/s	23	Pulse	Requires	10 m
11	Gas	Requires	Bluetooth	24	Gas	Requires	2 m
12	Pulse	Requires	5 Mb/s	25	Gas	Requires	2 s
13	Gas	Requires	ZigBee				

different communication protocols where each protocol can ensure a different security level. A similar situation is with other action modules, as well.

Therefore, from 120 configurations which are presented by the specialized feature model, we must select one. All these configurations are understood as a design space of the IoT-based healthcare application's BAN layer. The selected configuration from the design space should present the IoT-based healthcare BAN application best. We used several criteria for the selection of a configuration; these criteria present QoS requirements for the IoT-based healthcare application which were not described in the application specialization. Moreover, from the selected configuration code framework, an application which controls action and communication modules of the case study application will be generated in the generation phase of the proposed IoT-based application development method.

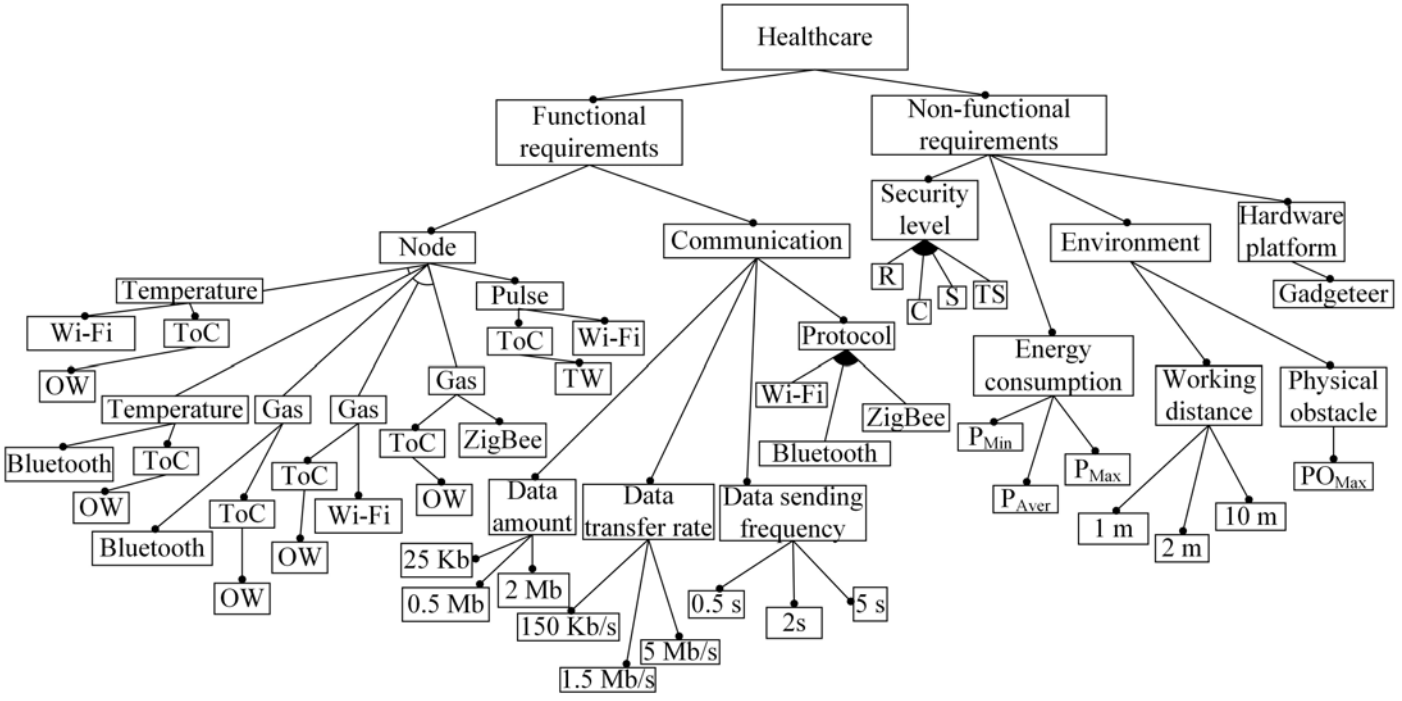


Figure 4.5. Specialized feature model of IoT-based healthcare application

Table 9. Specialized feature model characteristics

No.	Data	Specialized feature model
	Statistics	
1	Features	64
1.1	-Optional	0
1.2	-Mandatory	51
1.3	- Grouped	12
1.4	- Groups	4
2	Tree Depth	6
3	Extra constraints	25
4	Distinct extra constraints variables	26
5	Clause Density	0.41
6	CNF Clauses	1.08
	Results	
1	Consistency	Consistent
2	Dead Features	0
3	Common Features	49
	Metrics	
1	Count Configurations	120
2	Variability Degree (%)	6.505E-18

For the selection of the best configuration when striving to implement the IoT-based healthcare application from which the code framework will be generated, we used multi- criteria optimization as a DSE process solution (see Chapter 3.5.2). To perform optimization, we used QoS requirements as optimization parameters and information from the components database (see Chapter 3.5.3).

In Figure 4.6, the DSE process is shown which was used to get the specialized configuration. As it can be seen, the DSE process takes the specialized feature model, components database and optimization parameters as inputs. At first, the DSE process takes the inputs and transmits them to the optimization algorithm. After the optimization has been completed, the DSE process generates the specialized configuration of the IoT-based healthcare BAN application as an output. This generated configuration describes the IoT-based healthcare application's BAN layer according to QoS requirements best. At this point, configuration is understood as a list of exact sensors, actuators, communication modules and these communication modules working modes which, basically, present a security level which must be used by the communication module for data transferring (see Figure 4.8).

As it can be seen from Figure 4.6, for the DSE process, a specialized feature model has been used. This model can be presented in graphical or textual formats. For the DSE process and the selection of the best configuration to

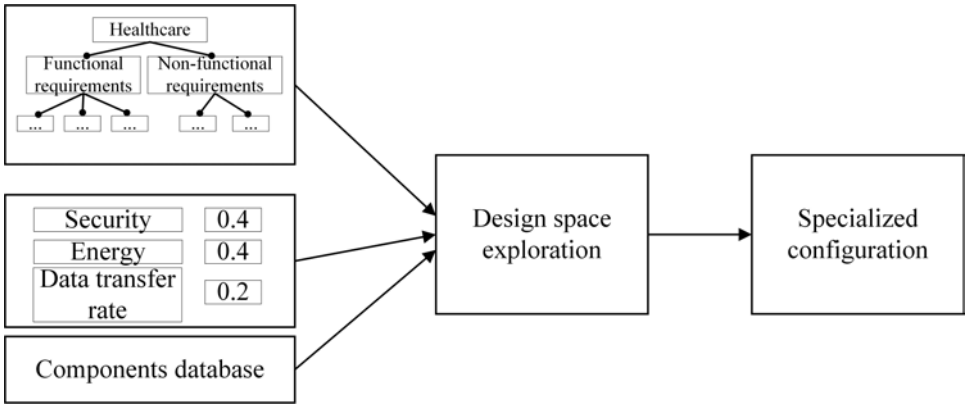


Figure 4.6. Design space exploration

implement the IoT-based healthcare application, we use the specialized feature model which is presented in the textual format; the SXFM format is used to present feature models in the textual format. In Chapter 3.5.2, we presented which features are used for multi-criteria optimization from the aggregated model and which information comes from the components database. As the DSE process uses the specialized feature model in the SXFM format to select the features which are used in optimization, the specialized feature model must be parsed. For parsing of the specialized feature model, we used the SMFM format parser library [1] presented by the S.P.L.O.T. tool creators

There are many QoS parameters which are important for the IoT-based healthcare application's BAN layer. These different parameters describe different performance of the application. Thus, for example, for the purpose we have chosen *energy*, *security* and *data transfer rate* as the main criteria of QoS, according to which, the configuration must be chosen and which, we think, are the most important for the IoT-based healthcare application. The selected QoS parameters indicate that the security level must be as high as possible with the absolutely minimum energy consumption. As it was mentioned in Chapter 3.5.2, for the optimization, we use the weighted sum method. This method requires that each parameter must have some assigned value, which is presented as a weight. For example, for the purpose, the weights for the QoS service were assigned as follows: security – 0.4, energy – 0.4 and the data transfer rate – 0.2.

As can be seen from Figure 4.6, in order to perform DSE, the components database must be used. As it was mentioned in Chapter 3.5.3, the components database contains information about the component characteristics, and some of these characteristics for communication modules must be calculated in order to perform the DSE more accurately. Therefore, in order to create the components database and to calculate the required character-

[1] http://ec2-52-32-1-180.us-west-2.compute.amazonaws.com:8080/SPLOT/sxfm.jar

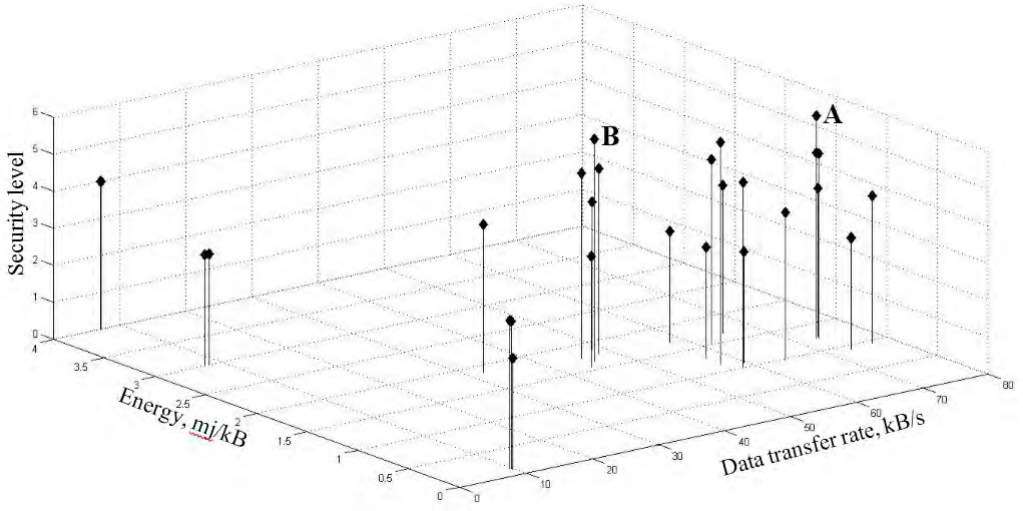


Figure 4.7. Design space exploration of the IoT-based healthcare application's BAN module

istics, we have used a methodology presented in [Venckauskas et al., 2014a, Venckauskas et al., 2014b]. The following communication modules of Gadgeteer platform have been included into the components database: Wi-Fi RS21, Wi-Fi RN171, Bluetooth, XBee Pro and XBee (the latter two are used for ZigBee protocol); hereby characteristics of these communication modules have been calculated. Some of the calculated characteristics are presented in [Venckauskas et al., 2014a, Venckauskas et al., 2014b, Venckauskas et al., 2016a]. Also, Pulse Oximeter, Temp and GasSence sensors were included into the components data base, with their parameters (see 3.19). In order to evaluate the security levels, the following estimates were assigned: R-1, C-2, S-3, TS-4.

The design space (different PLs) intended to implement the IoT-based healthcare application is created when all the data has been passed to the design space exploration process. As it was mentioned earlier (see Chapter 3.5.2), the design space exploration process performs the evaluation of configurations which are presented in the specialized model according to the QoS requirements of applications and selects one which describes the performance of the application best. In our case, these requirements are security, energy and data transfer rate; due to this fact, the design space can be presented as a cube graph (see Figure 4.7) where each axis represents athe different requirement of QoS. Due to its readability, Figure 4.7 presents just a part of the design space. In Figure 4.7, each point presents a different configuration which can be used to implement the IoT application. At this point, each configuration presents a different set of hardware components with their characteristics as it is presented in

the components database (see Chapter 3.5.3), which can be used to implement nodes of the IoT-based healthcare application's BAN layer. In Figure 4.7, for example, *A* configuration presents the following components which can be used for the implementation of nodes of the IoT application (further on, only the names of components, without their characteristics are presented): Pulse node: Pulse_oximeter sensor, Wi-Fi_RS21 communication module, WEP_128 working mode, temperature node: Temp sensor, Bluetooth communication module, Mode3 working mode, Gas node: GasSence sensor, ZigBee communication module, ENC-MIC-64 working mode. *B* configuration: Pulse node: Pulse_oximeter sensor, Wi-Fi_RS21 communication module, WEP_128 working mode, temperature node: Temp sensor, Bluetooth communication module, Mode3 working mode, Gas node - GasSence sensor, Wi-Fi_RN171 communication module, WEP_128 working mode. In addition, components' names are presented as described in Chapter 3.5.3, and each component is described by the same characteristics as it is presented in Chapter 3.5.2. As it was mentioned in Chapter 3.5.2, these characteristics are used for the evaluation of configurations and the selection of the configuration which describes the QoS requirements best of all. Thus in order to evaluate these characteristics according to the QoS requirements, the Pareto optimality method is used (see Chapter 3.5.2).

In Figure 4.8, we show the generated specialized configuration which was created by using the DSE process (see Figure 4.6), the components database, security and power as QoS parameters, and the specialized feature model. This configuration will be used for the development of the IoT-based healthcare application. The generated specialized configuration is presented in XML format. As it can be seen from the generated configuration, for each node which is used in the IoT-based application development, specific action modules are assigned which should be used for the action performance of the node. From the specialized configuration, we can see that, for example, the *Pulse* node is assigned the *Pulse_oximeter* sensor. Also, as it can be evidently seen, for this sensor, the specific communication module has been assigned, this communication module is used for data transferring, also, the working mode of this communication module is presented. For the remaining nodes which are used for the development of the IoT-based healthcare application's BAN layer, similar information is assigned: the action module, the communication module, and the communication module's working mode derived from the components database and the optimization process while the type of communication comes from the specialized feature model.

Now, when we know which action and communication modules should be used for the development of each node of the IoT-based healthcare application's BAN layer. The real structure of the IoT-based healthcare application's BAN layer can be distinguished from the specialized configuration. In Figure 4.9, we present the BAN layer's structure, for which, a code framework will

```
<Healthcare>
 <Hardware platform> Gadgeteer</Hardware platform>
 <Pulse>
      <Action_module>Pulse_oximeter</Action_module>
           <Communication module_name = "WI-Fi_RS21" >
           <Security_level>WEP_128</Security_level>
           </Communication>
           <TypeOfCommunication>TW</TypeOfCommunication>
 </Pulse>
 <Temperature>
      <Action_module>Temp</Action_module>
           <Communication module_name = "Bluetooth" >
           <Security_level>Mode3</Security_level>
           </Communication>
           <TypeOfCommunication>OW</TypeOfCommunication>
 </Temperature>
  <Gas>
           <Action_module>Pulse_oximeter</Action_module>
           <Communication module_name = "ZigBee_XBee" >
           <Security_level>ENC-MIC-64</Security_level>
           </Communication>
           <TypeOfCommunication>OW</TypeOfCommunication>
 </Gas>
</Healthcare>
```

Figure 4.8. The selected specialized configuration IoT-based healthcare BAN module

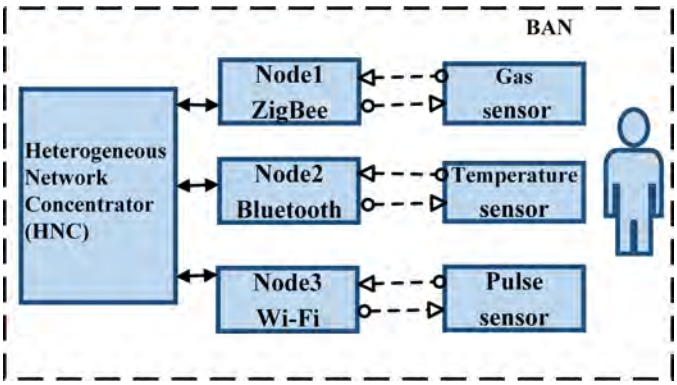


Figure 4.9. BAN structure after the specialization phase

be generated in the generation phase of the proposed IoT-based application development method.

4.2.5. Framework generation for IoT-based healthcare application's BAN layer

In this chapter, we present code framework generation from the optimized configuration. For code generation, we created a code generator whose work processes are presented in Figure 3.6. The code generator is based on code template libraries. Since a large part of the code generation is plain text matching, replacement and XML processing, a scripting language is more suitable for the creation of the code generator. For the implementation of the code generator, we used Python programming language. It is denoted by convenient libraries for text processing, file system operations, regular expression matching, and XML processing. In general, any programming language is suitable to implement the proposed code generator.

The code generator is designed to connect programming components from the code template library with hardware components which are presented in the specialized configuration. The code generator defines which code's template library must be used and how to link the source code according to the specialized configuration. The input for the code generator is the specialized configuration of the IoT-based healthcare application's BAN layer implementation as it is presented in Figure 4.8 and the code template library (see Figure C.1 in Appendix C); whereas the output of the generator is a generated code framework which will be used to implement the IoT-based healthcare BAN application. In Figure 4.10, we present the basic structure of code generation processes.

For the parsing of the specialized configuration, we used XML processing libraries which are implemented by Python programming language. By using our code generator, the XML file of the specialized configuration which was

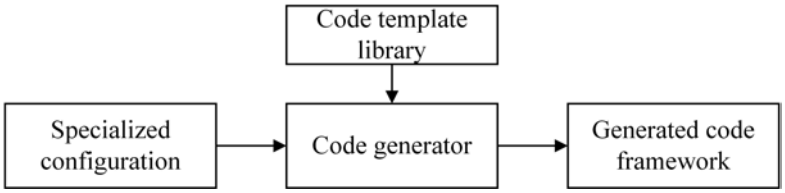


Figure 4.10. Generation of code framework

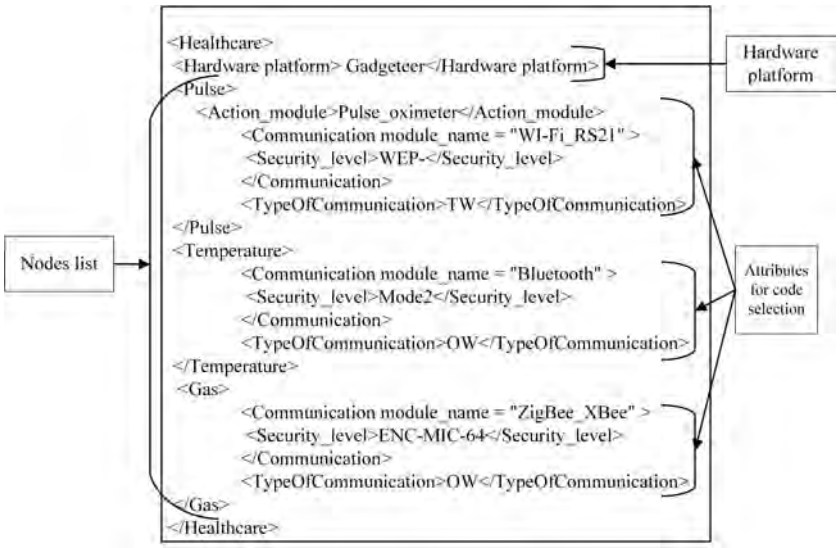


Figure 4.11. The parsing process of specialized configuration

created in the specialization phase is divided into two lists (see Figure 4.11 and Chapter 3.6): the nodes information and the information about the hardware platform. The additionally created node list is divided into attributes for each node (see Figure 4.11) whereas the attributes are used for code selection from the code template as it is presented in Chapter 3.6.

As it is presented in Chapter 3.6, the first step of the created code generator is the determination which hardware platform must be used for the implementation of the IoT-based healthcare application's BAN layer. By using this information, the necessary code template library from the code template repository is selected. Code template libraries are one of the core features of our proposed code generator. Therefore, for example, the purpose code template library was created. The code template library is created by using the rules described in Chapter 3.6.1. In Figure C.1 (Appendix C), we present a fragment of the created code template library. As it can be seen, this created code template library contains programming components of Gadgeteer platform which describe the performance of hardware components of this platform. Figure C.1

(Appendix C) presents only the source code which describes such Pulse sensor's actions as the pulse measurement performance and checking if the pulse sensor is correctly attached. The same is with other methods describing the components of the Gadgeteer platform.

The next step of the code generator after the necessary code template library is selected is the node information extraction from the specialized configuration. From Figure 4.11, it can be seen that for the implementation of the BAN layer of the IoT-based healthcare application, three nodes are used where each node's description contains attributes which are used for code generation. For each node, the source code is generated separately. As it was mentioned in Chapter 3.6, the attributes of code generation and the source code are linked through the names of modules which are presented in a specialized configuration and the tag name of the code template library. During the code generation process, all the source codes which describe the node performance are selected. For example, during the code generation for Pulse node, all the methods describing the Pulse_oximeter, and Wi-Fi_RS21 module's performance and two-way data transferring for the Wi-Fi protocol are selected.

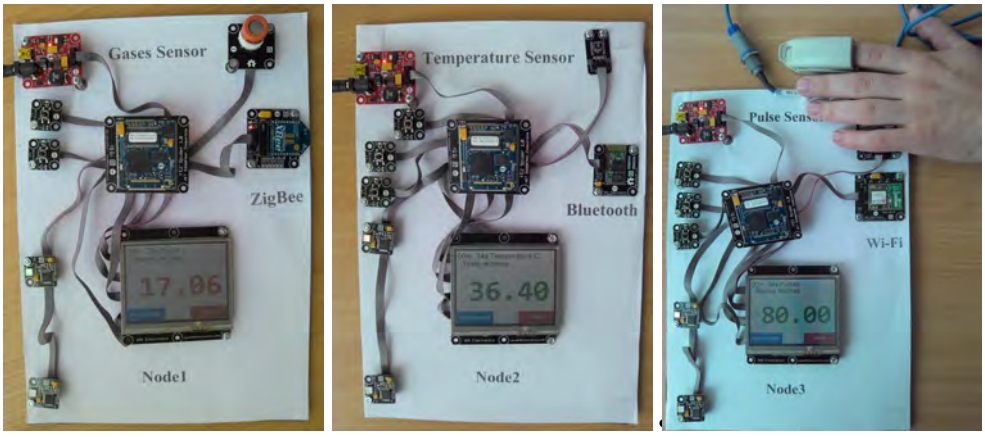
Due to the fact that code generator, search source code for each node, which is used to implement IoT-based healthcare application's BAN layer, separately, we get three different lists of methods, where each is created for specific node according attributes presented in the specialized configuration. Methods in these sets can repeat oneself, because the same methods can be used by different nodes for implementation. Such methods can describe the performance of communication module or data transfer. Figure C.3 (Appendix C), presents three lists of methods for each application node, which were done by code generator using created code template library and code generation information. Presented code sets show only method names, without source code, which were selected from code template library.

The last step of the created code generator is to determine the output format of the selected code and output file extension. In the case study, the code selected from the code template library must be exported by using classes as the Gadgeteer platform requires an object-oriented programming language. This information comes from the code template library. In Figure C.2 (Appendix C), we show the selected code's transformation from the methods list to the class for the Pulse node. Figure C.2 (Appendix C) presents just a fragment of the generated class. The same transformation is performed with other lists, as well. All the created classes in the case study (three classes in total) are exported in different files having the *.cs* extension where the file name is the same as the class name.

These three generated classes create a code framework which is used for the implementation of the BAN layer of the IoT-based healthcare application whose structure is shown in Figure 4.9.

4.2.6. Implementation of IoT-based healthcare application's BAN layer

The implementation phase is the last phase of the proposed IoT-based application's implementation method. This phase is used for the creation of a fully functional application by extending the generated IoT-based application framework with the necessary methods and logical connections. In our case, the fully functional application is an application which performs pulse, temperature and gas concentration measurements and sends the obtained data to the heterogeneous concentrator. The concentrator gathers this data and sends it to the healthcare-oriented application module via standard Internet as it is shown in Figure 4.1.



(a) Gas sensor **(b)** Temperature sensor **(c)** Pulse sensor

Figure 4.12. Implemented hardware of all three sensors of IoT-based healthcare BAN application

In the case study, according to the specifications of the IoT-based healthcare application, the Gadgeteer hardware platform for application implementation must be used. Therefore, we used Microsoft Visual Studio tools for the IoT-based healthcare application implementation from the generated framework as these tools offer opportunities to program the Gadgeteer hardware platform and support C# programming language in which the IoT-based healthcare application framework is generated. The generated code framework consists of three different code files, where each one contains a different class. These different classes describe different nodes of the IoT-based healthcare application's BAN layer. Therefore, for each class file, we created a project file where the generated class file served as a main; after that we implemented the generated framework to fully operating IoT-based healthcare application, until it matches the presented specifications (see Chapter 4.2.4). Thus we checked what programming components and logical links between them must be added. The

programming components which we added into the generated framework were as follows: namespaces, data sending intervals (which come from the specialized feature model, Figure 4.5), the IP address and port number of the remote server where the data must be sent (in this case of usage, it is heterogeneous concentrator, see Figure 4.1) and the methods are to be called at the right place. As it was mentioned before, this step of the implementation phase can be understood as verification of the generated code. After the implementation of the generated framework, we have fully functional body area application which performs temperature, pulse and gas measurements and sends the measurement data to the heterogeneous concentrator.

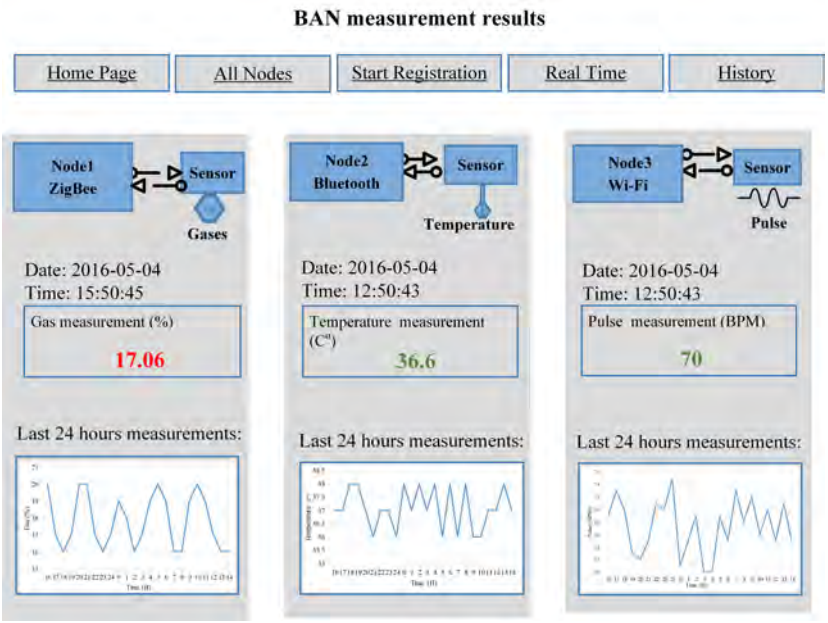


Figure 4.13. The view of real time measurements of the implemented BAN application

The nodes which are used for the IoT-based healthcare application's implementation consist of two main components: software and hardware. The software part was generated at the generation phase and extended until working on this. In order to obtain a fully working IoT-based healthcare application, we need Gadgeteer hardware components which must be used for the application's implementation and where the created software part would be uploaded. We may wonder which Gadgeteer hardware platform components must be taken are presented in the specialized configuration (see Figure 4.8). By using these components, we implemented the hardware part of each node which was presented in the specialized optimized configuration. The implemented hardware components with the uploaded software are shown in Figure 4.12.

In order to ensure that the generated and extended framework is working properly, we created an experimental BAN application whose basic structure is presented in Figure 4.1. In order to implement the healthcare-oriented application module, we created and launched a web-based service which collects the measured data and presents it in real time by using a web browser. We used standard Internet to send data from the BAN application to the healthcare-oriented application module. For this to be carried out, we used a laptop as a heterogeneous concentrator which supports all the three (Wi-Fi, ZigBee and Bluetooth) communication protocols. The test results of the measured data are shown in Figure 4.13.

4.3. Discussion

Below we discuss why we believe that the proposed IoT application development method can be applied to other application developments in other domains and not only for the body area network by using the same objectives as for the creation of the BAN application. We may wonder why the proposed methods can ensure the re-usability of components and code generation and why it differs from other similar methods. In addition, the evaluation of QoS of IoT applications can be performed at an early stage of its design process.

First of all, by using feature-based modelling, the complexity of the IoT-based application and the variability (the product line) to implement the application in its domain can be presented. In general, the feature models enable to capture the essential attributes of the applications to be modelled and to express them through the features and their relationships in the development cycle as early as possible. As systems (such as the IoT) are indeed very complex, the feature models fit well to represent the initial requirements of the domain under consideration. Features are abstract entities and, therefore, by using the feature-based notation, it is possible to represent the domain attributes of the different levels of abstraction and to model the domain variability explicitly. Here, by 'domain' we mean a possible set of systems within the IoT applications. As it has been shown, even in the case of the introduced body area network application, we had 120 different configurations, each of which was represented as a feature model for further analysis and implementation. These configurations were getting through the model specialization which is model transformation which does not depend on the specific domain. Therefore, they can be used for various IoT-based application developments.

Second, we are able – already in the early design stages (requirements modelling) – to analyze the system at the highest level of abstraction and thus to understand its core functionality features, to introduce changes into its functionality, and to model changes not from the scratch but systematically while bearing reuse and automation in mind. It is possible to reason about the bottlenecks of the system to be mitigated and to exclude them with much less effort

and resources. Furthermore, we are able to collect a set of approved artefacts (models are syntactically correct for the explicit variability management) for future applications. That is so because the models are the value *per se*. They can be used in multiple cases (as the tested knowledge units for experimentation or decision making). This way, the re-usability of the created feature models is ensured.

As the feature models also have the textual representation in some language (in our case, SXFM), it makes the lower-level transformations possible. Next, we are able to achieve a high reuse extent through automation by using the available transformations and the proposed generation tool because used model transformations, code template libraries and the code generator are not specific to the application domain.

Most technologies used in the development of the proposed IoT application development methods are well understood and have been widely used in other similar approaches, such as [Riedel et al., 2010, Grace et al., 2016, Patel and Cassou, 2015]. Compared with them, the proposed method is different because it treats the IoT application as a combination of software and hardware. Thus it enables us to evaluate the QoS of IoT application and proposes specific hardware which should be used to implement the application and generates code framework for this hardware.

The feature models enable us to represent the requirements uniformly, despite their quite different nature (energy, security, environmental factors), which enables us to determine and evaluate QoS attributes of IoT-based applications despite its domain only being in the early stage of the application creation. From the QoS perspective, we can evaluate the given application through the required level of security [Pastore and Dulaney, 2006], the required level of energy and the required level of performance. In order to do that we use a components database, which allows not only the evaluation of QoS, but also suggests the hardware components which should be used for the application implementation which is independent from the application and its domain. The evaluation of QoS at an early stage of the application design ensures the possibility to suggest the best configuration of the IoT-based application implementation as it has been shown in the investigation of use.

As a result of the provided case study, we have also identified some limitations of the proposed IoT application development method. First, for the seamless integration of model-to-model and model-to-program transformations, the adequate tools should be compatible. That was not the case with the tools we used because the modelling and verification tools are experimental. Therefore, in order to close this incompatibility gap, human interaction and additional software were required (for transforming the aggregated feature model SXFM (the output of S.P.L.O.T.) into the specialized configuration XML format that is supported by the proposed code generator). Therefore, the choice of the

available tools is an issue. Even though the use of the more powerful tools can increase the automation level, the methodology we propose does not suffer from the capabilities of the adequate tools. The other issue is the code template repository because the amount of the generated code depends on the particularity of the code template libraries. If the particularity of the code template is small (i.e. very few methods are presented in it), application developers must complete a lot of coding manually. Therefore, the need for manual interactions decreases the level of automation. These are purely technical limitations. They restrict, to some extent, the experimental investigation, we were able to provide. However, even the restricted experiment has enabled us to achieve the aim of the research and approve the soundness of our approach.

5. CONCLUSIONS

In this thesis, we have studied the model-driven method for Internet of things applications development, which is characterized by the opportunity to evaluate the following challenges of such applications: security and privacy; energy-awareness; environmental factors; diversity of sensors, actuators, communication protocols.

The relevant solution as the response to the emerging challenges is the use of prototyping combined with modern model-driven methodologies in designing the systems. Due to these facts, we have proposed a multi-layered IoT-based application development method which uses feature models.

1. The proposed IoT applications development method based on feature models is used to present possible configurations (different product lines) which can be used for the development of the IoT application in the specific IoT domain. Thus, the final application can be customized to meet the specifications of specific requirements.

2. The proposed IoT applications development method based on feature models is used to present possible configurations (different product lines) which can be used for the development of the IoT application in the specific IoT domain. Thus, the final application can be customized to meet the specifications of specific requirements.

3. The proposed generic feature models (functional and non-functional requirements) allow us to present: the variability of configurations (different PLs) to implement the application in a specific domain and the complexity of IoT-based applications with respect to security and energy requirements, environmental factors and the heterogeneity of devices and communication protocols.

4. Narrowing the variability space (PLs) of the aggregated feature model through the model specialization enables us to adapt configurations to the needs of the developing application requirements. This results in the creation of the specialized feature model, which presents the design space of configurations (PLs) meeting the specifications of IoT application which can be used for the implementation of the specific IoT-based application.

5. The result of design space exploration presented by the specialized feature model is a Pareto optimal feature model which presents the application which should be developed best and is used to generate a framework of the IoT application.

6. The proposed method is different from the analyzed IoT applications development methods presenting the variability of configurations (PLs)

which can be used for the implementation of the specific IoT application, evaluating each configuration according to the QoS requirements of application and providing a single best configuration to implement the given application.

7. Feature modelling and model transformations used in the proposed method, are independent from the IoT applications. Thus, the created aggregated feature models can be reused to implement other applications in the same domain or similar applications in other domains by skipping the modelling layer which decreases the application development time.

8. The proposed generic functional and non-functional requirements feature models were used in the case study for the development of the IoT-based healthcare BAN application. The experiment results showed that the complexity of the BAN layer and configurations (PLs) variability to implement the IoT-based healthcare application's BAN layer could be presented by using the proposed generic models. In addition, the Pareto optimal model for IoT-based healthcare BAN application framework generation could be created by using the proposed feature models.

References

[Abbas and Yoon, 2015] Abbas, Z. and Yoon, W. (2015). A Survey on Energy Conserving Mechanisms for the Internet of Things: Wireless Networking Aspects. *Sensors*, 15(10):24818–24847.

[Acher, 2011] Acher, M. (2011). *Managing Multiple Feature Models: Foundations, Language and Applications*. PhD thesis, UNIVERSITÉDE NICE-SOPHIA ANTIPOLIS - UFR Sciences École Doctorale de Sciences et Technologies de l'Information et de la Communication (STIC).

[Acher et al., 2010a] Acher, M., Collet, P., Lahire, P., and France, R. (2010a). Comparing Approaches to Implement Feature Model Composition. In *6th European Conference on Modelling Foundations and Applications (ECMFA)*, volume LNCS, page 16. Springer.

[Acher et al., 2010b] Acher, M., Collet, P., Lahire, P., and France, R. (2010b). Composing Feature Models. *Lecture Notes in Computer Science*, pages 62–81.

[Acher et al., 2013] Acher, M., Collet, P., Lahire, P., and France, R. B. (2013). Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657–681.

[Alam and Noll, 2010] Alam, S. and Noll, J. (2010). A Semantic Enhanced Service Proxy Framework for Internet of Things. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 488–495.

[Almeida et al., 2015] Almeida, A., Bencomo, N., Batista, T., Cavalcante, E., and Dantas, F. (2015). Dynamic decision-making based on NFr for managing software variability and configuration selection. *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC'15*, pages 1376–1382.

[Alves et al., 2006] Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., and Lucena, C. (2006). Refactoring product lines. *Proceedings of the 5th international conference on Generative programming and component engineering - GPCE-06*, pages 201–210.

[Andrews, 1991] Andrews, G. R. (1991). Paradigms for Process Interaction in Distributed Programs. *ACM Comput. Surv.*, 23(1):49–90.

[Anon et al., 2014] Anon, F., Navarathinarasah, V., Hoang, M., and Lung, C. H. (2014). Building a Framework for Internet of Things and Cloud Computing. In *2014 IEEE International Conference on Internet of Things*

(iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), pages 132–139.

[Apel and Kästner, 2009] Apel, S. and Kästner, C. (2009). An Overview of Feature-Oriented Software Development. *Journal of Object Technology*, 8(5):49–84. (column).

[Atzori et al., 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey . *Computer Networks*, 54(15):2787 – 2805.

[Ayala et al., 2015] Ayala, I., Amor, M., Fuentes, L., and Troya, J. M. (2015). A Software Product Line Process to Develop Agents for the IoT. *Sensors*, 15(7):15640–15660.

[Babar et al., 2010] Babar, S., Mahalle, P., Stango, A., Prasad, N., and Prasad, R. (2010). Proposed Security Model and Threat Taxonomy for the Internet of Things (IoT). *Communications in Computer and Information Science*, pages 420–429.

[Berger et al., 2013] Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wąsowski, A. (2013). A Survey of Variability Modeling in Industrial Practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 7:1–7:8, New York, NY, USA. ACM.

[Bhaddurgatte and Kumar, 2015] Bhaddurgatte, R. and Kumar, V. (2015). A Review: QoS Architecture and Implementations in IoT Environment. *Research & Reviews: Journal of Engineering and Technology*, pages 23–28.

[Brunet et al., 2006] Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., and Sabetzadeh, M. (2006). A manifesto for model merging. *Proceedings of the 2006 international workshop on Global integrated model management - GaMMa'06*.

[Capilla et al., 2013] Capilla, R., Bosch, J., and Kang, K.-C., editors (2013). *SYSTEMS AND SOFTWARE VARIABILITY MANAGEMENT*. Springer: Berlin, Germany.

[Chen et al., 2006] Chen, B., Wu, M., Yao, S., and Binbin, N. (2006). ZigBee Technology and Its Application on Wireless Meter-reading System. In *Industrial Informatics, 2006 IEEE International Conference on*, pages 1257–1260.

[Chen et al., 2014] Chen, S., Xu, H., Liu, D., Hu, B., and Wang, H. (2014). A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective. *IEEE Internet Things J.*, 1(4):349–359.

[Chen et al., 2009] Chen, X., Makki, K., Yen, K., and Pissinou, N. (2009). Sensor network security: a survey. *IEEE Communications Surveys Tutorials*, 11(2):52–73.

[ClauB and Jena, 2001] ClauB, M. and Jena, I. (2001). Modeling variability with UML. In *In GCSE 2001Young Researchers Workshop*.

[CoIoToTPC, 2013] CoIoToTPC (2013). Conclusions of the Internet of Things public consultation. Technical report, EUROPEAN COMMISSION.

[Conejar and Kim, 2016] Conejar, R. J. and Kim, H.-K. (2016). Conceptual Framework for Mobile Device Product Line Security based on Internet of Thing. *International Journal of Hybrid Information Technology*, 9(7):411–418.

[Czarnecki et al., 2005a] Czarnecki, K., Antkiewicz, M., Kim, C. H. P., Lau, S., and Pietroszek, K. (2005a). Model-driven Software Product Lines. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 126–127, New York, NY, USA. ACM.

[Czarnecki et al., 2012] Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., and Wąsowski, A. (2012). Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 173–182, New York, NY, USA. ACM.

[Czarnecki and Helsen, 2003] Czarnecki, K. and Helsen, S. (2003). Classification of Model Transformation Approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, pages 1–17.

[Czarnecki and Helsen, 2006] Czarnecki, K. and Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645.

[Czarnecki et al., 2005b] Czarnecki, K., Helsen, S., and Eisenecker, U. (2005b). Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29.

[Dalgarno, 2007] Dalgarno, M. (2007). Software Product Line Engineering with Feature Models. *Overload Journal*, 78:5–8.

[Davies, 2002] Davies, A. (2002). An overview of Bluetooth wireless technology and some competing LAN standards. *ICCSC'02. 1st IEEE International Conference on Circuits and Systems for Communications. Proceedings (IEEE Cat. No.02EX605)*, pages 206–211.

[de Fuentes et al., 2015] de Fuentes, J. M., Peris-Lopez, P., Tapiador, J. E., and Pastrana, S. (2015). Probabilistic yoking proofs for large scale IoT systems. *Ad Hoc Networks*, 32:43–52.

[Echeverria et al., 2015] Echeverria, J., Font, J., Pastor López, O., and Cetina Englada, C. (2015). Usability evaluation of variability modeling by means of common variability language. In *Complex Systems Informatics and Modeling Quarterly*, number 5, pages 61–81. RTU Press.

[Eisenhauer et al., 2010] Eisenhauer, M., Rosengren, P., and Antolin, P. (2010). HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems. *The Internet of Things*, pages 367–373.

[Filipe et al., 2015] Filipe, L., Fdez-Riverola, F., Costa, N., and Pereira, A. (2015). Wireless Body Area Networks for Healthcare Applications: Protocol Stack Review. *International Journal of Distributed Sensor Networks*, 2015:1–23.

[France and Rumpe, 2007] France, R. and Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA. IEEE Computer Society.

[Franky and Pavlich-Mariscal, 2012] Franky, M. C. and Pavlich-Mariscal, J. A. (2012). Improving implementation of code generators: A regular-expression approach. In *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, pages 1–10.

[Friedman et al., 2011] Friedman, R., Kogan, A., and Krivolapov, Y. (2011). On power and throughput tradeoffs of WiFi and Bluetooth in smartphones. In *INFOCOM, 2011 Proceedings IEEE*, pages 900–908.

[Fuentes et al., 2009] Fuentes, L., Nebrera, C., and Sanchez, P. (2009). Feature-Oriented Model-Driven Software Product Lines: The TENTE approach. In *Proceedings of the Forum of the 21st International Conference on Advanced Information Systems (CAiSE)*, volume 453, pages 67–72.

[Fukui et al., 2013] Fukui, T., Matsuura, S., Inomata, A., and Fujikawa, K. (2013). A Two-tier Overlay Publish/Subscribe System for Sensor Data Stream Using Geographic Based Load Balancing. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 749–756.

[García et al., 2014] García, C. G., Espada, J. P., Valdez, E. R. N., and Díaz, V. G. (2014). Midgar: Domain-Specific Language to Generate Smart Objects

for an Internet of Things Platform. *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 352–357.

[Garcia-Dominguez and Kolovos, 2016] Garcia-Dominguez, A. and Kolovos, D. S. (2016). Models from code or code as a model? In *16th International Workshop in OCL and Textual Modeling,*.

[Gass and Saaty, 1955] Gass, S. and Saaty, T. (1955). The computational algorithm for the parametric objective function. *Naval Research Logistics*, 2(1-2):39–45.

[Ge and Whitehead Jr., 2008] Ge, G. and Whitehead Jr., E. J. (2008). Rhizome: A Feature Modeling and Generation Platform. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 375–378.

[Glass, 2001] Glass, R. L. (2001). Frequently forgotten fundamental facts about software engineering. *IEEE Software*, 18(3):112–111.

[Grace et al., 2016] Grace, P., Pickering, B., and Surridge, M. (2016). Model-driven interoperability: engineering heterogeneous IoT systems. *Annals of Telecommunications*, 71:141–150.

[Greenfield and Short, 2003] Greenfield, J. and Short, K. (2003). Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, pages 16–27, New York, NY, USA. ACM.

[Griss et al., 1998] Griss, M. L., Favaro, J., and Alessandro, M. d. (1998). Integrating feature modeling with the RSEB. In *Proceedings of the 5th International Conference on Software Reuse*, ICSR '98, pages 76–85, Washington, DC, USA. IEEE Computer Society.

[Guan et al., 2006] Guan, Y., Ghose, A., and Lu, Z. (2006). Using constraint hierarchies to support QoS-guided service composition. *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 743–752.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660.

[Hachem et al., 2011] Hachem, S., Teixeira, T., and Issarny, V. (2011). Ontologies for the Internet of Things. In *ACM/IFIP/USENIX 12th International Middleware Conference*, page 3, Lisbon, Portugal. Springer.

[Han et al., 2013] Han, C., Jornet, J. M., Fadel, E., and Akyildiz, I. F. (2013). A cross-layer communication module for the Internet of Things. *Computer Networks*, 57(3):622–633.

[Haugen, 2012] Haugen, Ø. (2012). Common Variability Language (CVL) - OMG revised submission.

[Heer et al., 2011] Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S. L., Kumar, S. S., and Wehrle, K. (2011). Security Challenges in the IP-based Internet of Things. *Wirel. Pers. Commun.*, 61(3):527–542.

[Hemel et al., 2009] Hemel, Z., Kats, L. C. L., Groenewegen, D. M., and Visser, E. (2009). Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling*, 9(3):375–402.

[Hong-You and San-Ping, 2012] Hong-You, W. and San-Ping, Z. (2012). The Predigest Project of TCP/IP Protocol Communication System Based on {DSP} Technology and Ethernet. *Physics Procedia*, 25:1253 – 1257. International Conference on Solid State Devices and Materials Science, April 1-2, 2012, Macao.

[Honkanen et al., 2004] Honkanen, M., Lappetelainen, A., and Kivekas, K. (2004). Low end extension for Bluetooth. In *Radio and Wireless Conference, 2004 IEEE*, pages 199–202.

[Hu, 2015] Hu, P. (2015). A System Architecture for Software-Defined Industrial Internet of Things. *2015 IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB)*, pages 1–5.

[Huang et al., 2013] Huang, X., Li, Y., and Jin, S. (2013). A control system based on data exchange using ethernet and CANBUS for deep water AUV. In *Control Conference (ASCC), 2013 9th Asian*, pages 1–5.

[Hussain et al., 2015] Hussain, A., Wenbi, R., da Silva, A. L., Nadher, M., and Mudhish, M. (2015). Health and emergency-care platform for the elderly and disabled people in the smart city. *Journal of Systems and Software*, 110:253–263.

[Iec, 1995] Iec, I. (1995). Open Distributed Processing- Reference Model - Part 2: Foundations International Standard 10746-2 Itu-T Recommendation X.902.

[IERC, 2009] IERC (2009). Internet of Things Strategic Research Roadmap. Technical report, European Research Cluster on the Internet of Things.

[INFSO, 2008] INFSO (2008). Internet of Things in 2020, Roadmap for the Future. Technical report, INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems, In: Co-operation with the Working Group of European technology platform onsmart systems integration (EPoSS).

[ITU, 2005] ITU (2005). The Internet of Things. Technical report, International Telecumunication Union.

[ITU-T, 2005] ITU-T (2005). ITU Strategy and Policy Unit (SPU) ITU Internet Reports 2005: The Internet of Things. Technical report, International Telecommunication Union (ITU).

[ITU-T, 2012] ITU-T (2012). Overview of the Internet of things. Technical report, International Telecommunication Union.

[Jakob and Blume, 2014] Jakob, W. and Blume, C. (2014). Pareto Optimization or Cascaded Weighted Sum: A Comparison of Concepts. *Algorithms*, 7(1):166–185.

[Jin et al., 2012] Jin, J., Gubbi, J., Luo, T., and Palaniswami, M. (2012). Network architecture and QoS issues in the internet of things for a smart city. In *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, pages 956–961.

[Jouault et al., 2008] Jouault, F., Allilaire, F., Bezivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39.

[Jouault and Bezivin, 2006] Jouault, F. and Bezivin, J. (2006). KM3: A DSL for Metamodel Specification. In *Proceedings of the 8th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems*, FMOODS'06, pages 171–185, Berlin, Heidelberg. Springer-Verlag.

[Jusas et al., 2016] Jusas, N., Venckauskas, A., and Stuikys, V. (2016). Model driven framework to develop the IoT-based healthcare applications. In Papanikos, G. T., editor, *12th annual international conference on information technology and computer science, 16-19 May 2016, Athens, Greece : abstract book*, page 19. Athens Institute for Education and Research.

[Kalnins et al., 2010] Kalnins, A., Kalnina, E., Celms, E., and Sostaks, A. (2010). A model-driven path from requirements to code. *Computer Science and Information Technologies*, 756:33–57.

[Kamel Boulos and Al-Shorbaji, 2014] Kamel Boulos, M. N. and Al-Shorbaji, N. M. (2014). On the Internet of Things, smart cities and the WHO Healthy Cities. *Int J Health Geogr*, 13(1):10.

[Kamyabpour and Hoang, 2010] Kamyabpour, N. and Hoang, D. B. (2010). Modeling Overall Energy Consumption in Wireless Sensor Networks. In *Proceedings of the 2010 International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '10, pages 273–279, Washington, DC, USA. IEEE Computer Society.

[Kang et al., 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

[Kang et al., 1998] Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168.

[Karimpour and Ruhe, 2017] Karimpour, R. and Ruhe, G. (2017). Evolutionary robust optimization for software product line scoping: An explorative study . *Computer Languages, Systems & Structures*, 47, Part 2:189 – 210.

[Khanduri and S. Rattan, 2013] Khanduri, R. and S. Rattan, S. (2013). Performance Comparison Analysis between IEEE 802. 11a/b/g/n Standards. *IJCA*, 78(1):13–20.

[Kim et al., 2014] Kim, J., Lee, J., Kim, J., and Yun, J. (2014). M2M Service Platforms: Survey, Issues, and Enabling Technologies. *IEEE Communications Surveys Tutorials*, 16(1):61–76.

[Kotha and Pine, 1994] Kotha, S. and Pine, B. J. (1994). Mass Customization: The New Frontier in Business Competition. *The Academy of Management Review*, 19(3):588–592.

[Labiod et al., 2007] Labiod, H., Hossam, A., and Santis, C. D. (2007). *Wi-Fi, Bluetooth, Zigbee and WiMax*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Lanzisera et al., 2014] Lanzisera, S., Weber, A. R., Liao, A., Pajak, D., and Meier, A. K. (2014). Communicating Power Supplies: Bringing the Internet to the Ubiquitous Energy Gateways of Electronic Devices. *IEEE Internet of Things Journal*, 1(2):153–160.

[Lee et al., 2013] Lee, G. M., Crespi, N., Choi, J. K., and Boussard, M. (2013). Internet of Things. *Lecture Notes in Computer Science*, pages 257–282.

[Lee et al., 2002] Lee, K., Kang, K. C., and Lee, J. (2002). *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*, pages 62–77. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Li et al., 2014] Li, L., Rong, M., and Zhang, G. (2014). An Internet of things QoS estimate approach based on multi-dimension QoS. In *Computer Science Education (ICCSE), 2014 9th International Conference on*, pages 998–1002.

[Li et al., 2015] Li, L., Rong, M., and Zhang, G. (2015). An Internet of Things QoE evaluation method based on multiple linear regression analysis. In *Computer Science Education (ICCSE), 2015 10th International Conference on*, pages 925–928.

[Li and Tang, 2014] Li, L. and Tang, S. (2014). Some Reform Ideas for the Software Project Management Course. *Proceedings of the 3rd International Conference on Science and Social Research*.

[Libelium, 2014] Libelium (2014). Bluetooth Low Energy Networking Guide. Technical report, Libelium Comunicaciones Distribuidas S.L.

[Lockheed and U.S. Navy, 2013] Lockheed, M. and U.S. Navy (2013). A Glossary of Product Line Engineering. Technical report, ;BigLever Software Inc.: Austin, TX, USA,.

[Lopez et al., 2014] Lopez, L., Ozdemir, O., Kuemper, D., Stanca-Kaposta, B., Chainho, P., De, S., and Sasu, L.-M. (2014). Internet of Things Environment for Service Creation and Testing. Technical report, European Commission.

[Maazoun et al., 2016] Maazoun, J., Bouassida, N., and Abdallah, H. B. (2016). Variability modeling with a SPL-UML profile. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 201–207.

[MacDonald et al., 2005] MacDonald, A., Russell, D., and Atchison, B. (2005). Model-driven development within a legacy system: an industry experience report. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, pages 14–22.

[Mahalank et al., 2016] Mahalank, S. N., Malagund, K. B., and Banakar, R. M. (2016). Design space exploration for IoT based traffic density indication system. In *2016 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1–6.

[Marler and Arora, 2004] Marler, R. and Arora, J. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395.

[Mattern and Floerkemeier, 2010] Mattern, F. and Floerkemeier, C. (2010). From Active Data Management to Event-based Systems and More. chapter From the Internet of Computers to the Internet of Things, pages 242–259. Springer-Verlag, Berlin, Heidelberg.

[Memon et al., 2014] Memon, M., Wagner, S., Pedersen, C., Beevi, F., and Hansen, F. (2014). Ambient Assisted Living Healthcare Frameworks, Platforms, Standards, and Quality Attributes. *Sensors*, 14(3):4312–4341.

[Mendonca et al., 2009] Mendonca, M., Branco, M., and Cowan, D. (2009). S.P.L.O.T. - Software product lines online tools. *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA09*, pages 761–762.

[Mernik et al., 2005] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and How to Develop Domain-specific Languages. *ACM Comput. Surv.*, 37(4):316–344.

[Miao and Siek, 2014] Miao, W. and Siek, J. (2014). Compile-time Reflection and Metaprogramming for Java. In *Proceedings of the ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation*, PEPM '14, pages 27–37, New York, NY, USA. ACM.

[Mukherjee et al., 2014] Mukherjee, S., Dolui, K., and Datta, S. K. (2014). Patient health management system using e-health monitoring architecture. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 400–405.

[Nanevski, 2002] Nanevski, A. (2002). Meta-programming with Names and Necessity. *SIGPLAN Not.*, 37(9):206–217.

[Neufeld and Goldberg, 1990] Neufeld, G. W. and Goldberg, M. W. (1990). A request/response protocol for ISO remote operations. In *Computer and Communication Systems, 1990. IEEE TENCON'90., 1990 IEEE Region 10 Conference on*, pages 623–627 vol.2.

[Nikiforova et al., 2009] Nikiforova, O., Cernickins, A., and Pavlova, N. (2009). Discussing the Difference between Model Driven Architecture and Model Driven Development in the Context of Supporting Tools. In *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, pages 446–451.

[Pastore and Dulaney, 2006] Pastore, M. and Dulaney, E. (2006). *CompTIA Security+ Deluxe Study Guide*. John Willey & Sons.

[Patel and Cassou, 2015] Patel, P. and Cassou, D. (2015). Enabling high-level application development for the Internet of Things. *Journal of Systems and Software*, 103:62–84.

[Patel et al., 2013] Patel, P., Kattepur, A., Cassou, D., and Bouloukakis, G. (2013). Evaluating the Ease of Application Development for the Internet of Things. Technical report.

[Patel et al., 2015] Patel, P., Luo, T., and Bellur, U. (2015). Evaluating a Development Framework for Engineering Internet of Things Applications. *arXiv preprint arXiv:1606.02119*.

[Patel et al., 2011] Patel, P., Pathak, A., Teixeira, T., and Issarny, V. (2011). Towards application development for the Internet of Things. In *ACM/IFIP/USENIX 12th International Middleware Conference*, page 5, Lisboa, Portugal.

[Peng and Ruan, 2012] Peng, D. and Ruan, Y. (2012). AHP-based QoS Evaluation Model in the Internet of Things. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on*, pages 578–581.

[Poruban et al., 2014] Poruban, J., Bacikova, M., Chodarev, S., and Nosal, M. (2014). Pragmatic model-driven software development from the viewpoint of a programmer: Teaching experience. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1647–1656.

[Riebisch et al., 2004] Riebisch, M., Streitferdt, D., and Pashov, I. (2004). Modeling Variability for Object-Oriented Product Lines. *Lecture Notes in Computer Science*, pages 165–178.

[Riedel et al., 2010] Riedel, T., Yordanov, D., Fantana, N., Scholz, M., and Decker, C. (2010). A model driven Internet of Things. In *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, pages 265–268.

[Roman et al., 2013] Roman, R., Zhou, J., and Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things . *Computer Networks*, 57(10):2266 – 2279. Towards a Science of Cyber SecuritySecurity and Identity Architecture for the Future Internet.

[Rugaber and Stirewalt, 2004] Rugaber, S. and Stirewalt, K. (2004). Model-driven reverse engineering. *IEEE Software*, 21(4):45–53.

[Saha et al., 2017] Saha, H. N., Mandal, A., and Sinha, A. (2017). Recent trends in the Internet of Things. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–4.

[Salihbegovic et al., 2015] Salihbegovic, A., Eterovic, T., Kaljic, E., and Ribic, S. (2015). Design of a domain specific language and IDE for internet of things applications. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, pages 996–1001.

[Sallai, 2014] Sallai, G. (2014). Future Internet Visions and Research Clusters. *Acta Polytechnica Hungarica*, 11(7):5–24.

[Sancho, 2009] Sancho (2009). Definition for the term (software) service, sector abbreviations and definitions for a telecommunications thesaurus oriented database. Technical report, ITU-T.

[Schmidt et al., 2007] Schmidt, D., Kramer, M., Kuhn, T., and Wehn, N. (2007). Energy modelling in sensor networks. *Advances in Radio Science*, 5:347–351.

[Shaoshuai et al., 2011] Shaoshuai, F., Wenxiao, S., Nan, W., and Yan, L. (2011). MODM-Based Evaluation Model of Service Quality in the Internet of Things. *Procedia Environmental Sciences*, 11:63–69.

[Shoshani et al., 2010] Shoshani, G., Mitschke, S., and Stephan, S. (2010). Industrial Fieldbus technology and Fieldbus cable overview - Cable standards and electrical qualifications. In *Petroleum and Chemical Industry Conference (PCIC), 2010 Record of Conference Papers Industry Applications Society 57th Annual*, pages 1–10.

[Skarmeta et al., 2014] Skarmeta, A. F., Hernandez-Ramos, J. L., and Moreno, M. V. (2014). A decentralized approach for security and privacy challenges in the Internet of Things. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 67–72.

[Slavin et al., 2014] Slavin, R., Lehker, J. M., Niu, J., and Breaux, T. D. (2014). Managing security requirements patterns using feature diagram hierarchies. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 193–202.

[Smith, 2012] Smith, I. (2012). The Internet of Things 2012: New Horizons. Technical report, European Research Cluster on the internet of thongs.

[Streitferdt et al., 2003] Streitferdt, D., Riebisch, M., and Philippow, K. (2003). Details of formalized relations in feature models using OCL. In *Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the*, pages 297–304.

[Stuikys and Damasevicius, 2013] Stuikys, V. and Damasevicius, R. (2013). Meta-programming and model-driven meta-program development. *Advanced Information and Knowledge Processing*.

[Sun et al., 2008] Sun, Y., Demirezen, Z., Mernik, M., Gray, J., and Bryant, B. (2008). Is my DSL a modeling or programming language. In *in: Proceedings of 2nd International Workshop on Domain-Specific Program Development (DSPD)*, pages 1–4.

[Swamy et al., 1995] Swamy, S., Molin, A., and Covnot, B. (1995). OO-VHDL. object-oriented extensions to VHDL. *Computer*, 28(10):18–26.

[Tabish et al., 2013] Tabish, R., Mnaouer, A. B., Touati, F., and Ghaleb, A. M. (2013). A comparative analysis of BLE and 6LoWPAN for U-HealthCare applications. In *2013 7th IEEE GCC Conference and Exhibition (GCC)*, pages 286–291.

[Teixeira et al., 2011] Teixeira, T., Hachem, S., Issarny, V., and Georgantas, N. (2011). Service Oriented Middleware for the Internet of Things: A Perspective. In *Proceedings of the 4th European Conference on Towards a Service-based Internet*, ServiceWave'11, pages 220–229, Berlin, Heidelberg. Springer-Verlag.

[Thibault and Consel, 1997] Thibault, S. and Consel, C. (1997). A framework for application generator design. *Proceedings of the 1997 symposium on Software reusability - SSR '97*, 22(3):131–135.

[Thum et al., 2009] Thum, T., Batory, D., and Kastner, C. (2009). Reasoning about edits to feature models. *2009 IEEE 31st International Conference on Software Engineering*, (254–264).

[Vanderperren and Mueller, 2006] Vanderperren, Y. and Mueller, W. (2006). UML and model-driven development for SoC design. In *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, pages 1–1.

[Venckauskas et al., 2014a] Venckauskas, A., Jusas, N., Kazanavicius, E., and Stuikys, V. (2014a). Identification of Dependency among Energy Consumption and Wi-Fi Protocol Security Levels within the Prototype Module for the IoT. *ElAEE*, 20(6):132–135.

[Venckauskas et al., 2014b] Venckauskas, A., Jusas, N., Toldinas, J., and Kazanavicius, E. (2014b). Security Level versus Energy Consumption in Wireless Protocols for Internet of Things. *Information and Software Technologies*, pages 419–429.

[Venckauskas et al., 2016a] Venckauskas, A., Stuikys, V., Damasevicius, R., and Jusas, N. (2016a). Modelling of Internet of Things units for estimating security-energy-performance relationships for quality of service and environment awareness. *Security and Communication Networks*, 9(16):3324–3339.

[Venckauskas et al., 2016b] Venckauskas, A., Stuikys, V., Jusas, N., and Burbaite, R. (2016b). Model-Driven Approach for Body Area Network Application Development. *Sensors*, 16(5):670.

[Venckauskas et al., 2016c] Venckauskas, A., Stuikys, V., Toldinas, J., and Jusas, N. (2016c). A Model-Driven Framework to Develop Personalized Health Monitoring. *Symmetry*, 8(7):65.

[Vermesan and Friess, 2014] Vermesan, O. and Friess, P., editors (2014). *Internet of Things - From Research and Innovation to Market Deployment*. River Publishers.

[Vermesan et al., 2013] Vermesan, O., Friess, P., Guillemin, P., Sundmaeker, H., Eisenhauer, M., and Moessner, K. (2013). *Internet of Things - Converging Technologies for Smart Enviroments ans Integrated Ecosystems*, chapter Internet of Things Strategic Research and Inovation Agenda, pages 7–152. River Publishers.

[Weber, 2010] Weber, R. H. (2010). Internet of Things - New security and privacy challenges. *Computer Law & Security Review*, 26(1):23–30.

[Yassein et al., 2016] Yassein, M. B., Mardini, W., and Khalil, A. (2016). Smart homes automation using Z-wave protocol. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–6.

[Zachariah et al., 2015] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. (2015). The Internet of Things Has a Gateway Problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 27–32, New York, NY, USA. ACM.

[Zheng et al., 2014] Zheng, X., Martin, P., Brohman, K., and Xu, L. D. (2014). Cloud Service Negotiation in Internet of Things Environment: A Mixed Approach. *IEEE Transactions on Industrial Informatics*, 10(2):1506–1515.

[Zhou et al., 2011] Zhou, H.-Y., Luo, D.-Y., Gao, Y., and Zuo, D.-C. (2011). Modeling of Node Energy Consumption for Wireless Sensor Networks. *WSN*, 03(01):18–23.

[Ziadi and Jézéquel, 2006] Ziadi, T. and Jézéquel, J.-M. (2006). Product Line Engineering with the UML: Deriving Products. In Pohl, K., editor, *Software Product Lines*, pages 557–588. Springer Verlag.

[Zionts, 1988] Zionts, S. (1988). Multiple Criteria Mathematical Programming: an Updated Overview and Several Approaches. *Mathematical Models for Decision Support*, pages 135–167.

PUBLICATIONS LIST BY THE AUTHOR

Indexed in the Web of Science with Impact Factor

1. Venčkauskas, Algimantas; **Jusas, Nerijus**; Kazanavičius, Egidijus; Štuikys, Vytautas. Identification of Dependency among energy consumption and Wi-Fi protocol security levels within the prototype module for the IoT // Elektronika ir elektrotechnika = Electronics and electrical engineering. Kaunas: KTU. ISSN 1392-1215. 2014, Vol. 20, no. 6, p. 132-135. [Science Citation Index Expanded (Web of Science); Inspec; Computers & Applied Sciences Complete; Central & Eastern European Academic Source] [Sc. fields: 01T]. [Contribution: 0.250]. [IF (E): 0.561 (2014)]
2. Venčkauskas, Algimantas; **Jusas, Nerijus**; Kazanavičius, Egidijus; Štuikys, Vytautas. An energy efficient protocol for the Internet of Things // Journal of electrical engineering-Elektrotechnický časopis. Berlin: De Gruyter. ISSN 1335-3632. 2015, vol. 66, iss. 1, p. 47-52. [Science Citation Index Expanded (Web of Science); Scopus; Inspec] [Sc. fields: 07T]. [Contribution: 0.250]. [IF (E): 0.407 (2015)]
3. Venčkauskas, Algimantas; Štuikys, Vytautas; **Jusas, Nerijus**; Burbaitė, Renata. Model-driven approach for body area network application development // Sensors. Basel: MDPI AG. ISSN 1424-8220. 2016, vol. 16, iss. 5, Article 670, p. [1-22]. [Science Citation Index Expanded (Web of Science); Academic Search Complete] [Sc. fields: 07T]. [Contribution: 0.250]. [IF (E): 2.033 (2015)]
4. Venčkauskas, Algimantas; Štuikys, Vytautas; Toldinas, Jevgenijus; **Jusas, Nerijus**. A model-driven framework to develop personalized health monitoring // Symmetry. Basel: MDPI AG. ISSN 2073-8994. 2016, vol. 8, iss. 7, article 65, p. [1-18]. [Science Citation Index Expanded; Current Contents (Physical, Chemical & Earth Sciences); Academic Search Alumni Edition; Academic Search Complete; Academic Search Elite; Academic Search Premier; Academic Search Research & Development] [Sc. fields: 07T]. [Contribution: 0.250]. [IF (E): 0.841 (2015)]
5. Venčkauskas, Algimantas; Štuikys, Vytautas; Damaševičius, Robertas; **Jusas, Nerijus**. Modelling of Internet of Things units for estimating security-energy-performance relationships for Quality of Service and environment awareness // Security and communication networks [electronics resource]. Hoboken, NJ: John Wiley & Sons. ISSN 1939-0114. 2016, vol. 9, iss. 16, p. 3324-3339. [Science Citation Index Expanded (Web of Science); Current Contents (Engineering, Computing & Technology)] [Sc. fields: 07T]. [Contribution: 0.250]. [IF (E): 0.806 (2015)]

Publications in other international databases

1. Venčkauskas, Algimantas; **Jusas, Nerijus**; Toldinas, Eugenijus; Kazanavičius, Egidijus. Security Level Versus Energy Consumption in Wireless Protocols for Internet of Things // Information and software technologies: 20th international conference, ICIST 2014, Druskininkai, Lithuania, October 9-10, 2014 : proceedings / Giedre Dregvaite; Robertas Damasevicius (eds.) ; Kaunas University of Technology. Cham: Springer, 2014. (Communications in computer and information science, 465, ISSN 1865-0929), ISBN

9783319119571. p. 419-429. [Conference Proceedings Citation Index; SpringerLINK] [Sc. fields: 07T]. [0.250]

OTHER PUBLICATIONS

1. **Jusas, Nerijus**; Venčkauskas, Algimantas; Štuikys, Vytautas. Model driven framework to Develop the IoT-based Healthcare Applications // 12th annual international conference on information technology and computer science, 16-19 May 2016, Athens, Greece : abstract book / edited by Gregory T. Papanikos. Athens: Athens Institute for Education and Research, ISBN 9789605980474. p. 24. [Sc. fields: 07T]. [0.333]

SL344. 2017-05-15, 17,75 leidyb. apsk. l. Tiražas 12 egz. Užsakymas 169.

Išleido Kauno technologijos universitetas, K. Donelaičio g. 73, 44249 Kaunas
Spausdino leidyklos „Technologija“ spaustuvė, Studentų g. 54, 51424 Kaunas

APPENDIX

A. DEFINITIONS OF MAIN CONCEPTS OF FEATURE MODEL

The following definitions are used to describe the feature models [Venckauskas et al., 2016c, Venckauskas et al., 2016a]:

Definition 1 *(Feature model). Feature model (FM) is the directed graph G(X, V). The latter represents the composition of the feature tree T(X, U) and a set of edges B specifying constraints between nodes* $X_b \subset X$ *where U is a set of directed edges (arcs) representing parent-child relationships among a pair of nodes;* $V = U \cup B$. *Feature model is a tuple* $\langle G(X,V), FM_{cons} \rangle$, FM_{cons} *is a list of feature model constraints (include and exclude).*

Definition 2 *(Feature). A feature is a node* $x_i (x_i \in X)$ *of the feature graph G(X,V) where X is a set of features, and V is set of edges that represent relationships among features.*

Definition 3 *(Root feature). The top-level feature that has no parents is the root feature. Graph G has only one root feature.*

Definition 4 *(Relationship between features) A relationship between features is a sub-graph* $G_r(X_r, U_r)$ *with properties (i)-(iv) as follows: (i)* $X_r = x_p \cup X_d$ *is a set of vertices, (ii)* $x_p \in X$ *is a parent vertex, (iii)* $X_d \subseteq X$ *is a set of vertices (grouped features) that are descendants (children) of* x_p, *(iv)* $U_r \subseteq U \subseteq V$ *is a set of edges that connect* x_p *to each member of* X_d.

Definition 5 *(Mandatory feature). Mandatory feature is such a feature that is always selected if its parents are selected.*

Definition 6 *(Optional feature). Optional feature is such a feature that may or may not be selected if its parents are selected.*

Definition 7 *(Varian point). A feature that is a parent of either an optional or an alternative feature group.*

Definition 8 *(Alternative feature, XOR). An alternative feature of the feature grouped features is the only one feature that must be selected if its parent is selected.*

Definition 9 *(OR). An optional feature of the grouped features is one or more features that may be selected if its parent is selected.*

Definition 10 *(Constraint). A constraint is a predicate of a prescribed type between two vertices-variants* x_i *and* x_j *in* G. *Formally,* $b_t : (x_i, x_j) \rightarrow \{true, false\}, b_t \in B; x_,x_j \in X$. *If the constraint exists, the predicate evaluates to true; otherwise the predicate evaluates to false.*

Definition 11 *(Requires constraint). A constraint indicating that the choice of one feature variant requires that another variant must be chosen is called require constrain. Formally, it is defined as* $b_{req}(x_i, x_j) \rightarrow true$.

Definition 12 *(Excludes constraint). A constraint indicating that the choice of one feature variant excludes another variant from being chosen is called exclude constraint. Formally, it looks like:* $b_{exc}(x_i, x_j) \rightarrow false$.

Definition 13 *(Dead feature). Feature f of FM is dead if it cannot be part of any of the valid configuration of FM.*

Definition 14 *(Core feature). Feature f of FM is a core if it is part of all the valid configuration of FM.*

Definition 15 *(Configuration). Feature model configuration is the model that contains all the mandatory nodes of the given feature model, may contain optional nodes, and includes variation points, but only one variant for each variation point is selected.*

Definition 16 *(Valid configuration). A valid configuration is the configuration that takes into account the constraints among variation points and variants.*

Definition 17 *Feature path is the path tree* $T^p (T^p \subset G)$ *that contains the only vertices selected by the analyser while travelling from the root to the selected leaves.*

Definition 18 *Feature path is the complete path. The complete feature path is formed from feature graph G when analysers make all the selections of the alternative and optional features.*

Definition 19 *Configuration K is a multi-set of all the features (vertices) in the feature path* T^p .

Definition 20 *Configuration K is a valid configuration if (i) it is not empty, (ii) it contains no variation points, (iii) the multiplicities (i.e. OR-relationships) of elements belonging to the multi-set are equal to 1, i.e., it contains only unique features, (iv) all the features in the multi-set satisfy a set of constraints B in the graph G.*

Definition 21 *(Correct feature model). A model that does not have dead feature and all configurations K are valid.*

Definition 22 *Feature model is said to be:*

i Abstract if some feature has no atomic features with concrete values or, in another context, some features may decompose into parts;

ii Aggregated if it consists of two or more abstract feature models where some feature has no atomic features with concrete values;

iii Specialized if it is derived from its ancestor feature model through removing some features (if a parent feature is removed, all of its children features are removed, too), and the atomic features have concrete values.

B. FEATURE MODEL EXAMPLE

Security methods are key parts of today's network applications. Security enables to ensure the secure data transmission from one part of an application to another or between different applications (secure communication, data encryption, and others). The security methods embedded in an application are determined by the application's features. Each distinct set of features defines a unique application in an application product line. Let us suppose that we only consider the features of encryption algorithms (symmetric, asymmetric and hybrid) and cypher key type (private and/or public). Figure B.1 represents a feature model of possibility to choose the necessary encryption algorithm and its parameters by using the basic feature model notation. Security methods (root feature, see Definition 3) consist of an encryption type, key type, security level and optionally a key exchange. A security level can be unprotected, protected and highly protected (choose one, XOR, see Definition 8), and the encryption algorithm is symmetric, asymmetric, hybrid, or all the possible options (OR, see Definition 9). As it can be seen from Figure B.1, any asymmetric feature requires a key exchange feature, while a symmetric feature excludes any public feature.

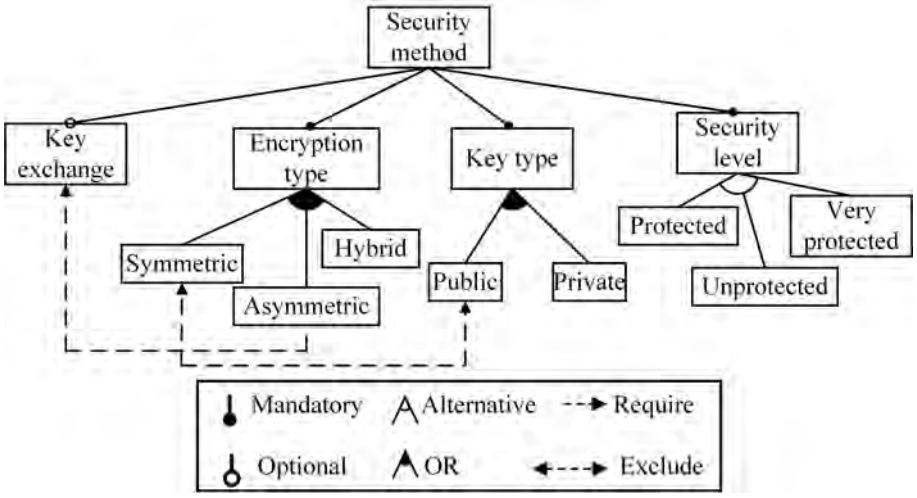


Figure B.1. Feature model of security methods

C. FRAGMENT OF CODE TEMPLATE AND OUTPUTS OF CODE GENERATOR

```
<platform name="Gadgeteer">
<meta>
<output file="cs"/>
<class name = "Yes"/>
</meta>
<code>
  <Pulse_Oximeter >
       bool IsConnected = null;
      void pulse_Oximeter_ProbeDetached(PulseOximeter
      sender)
         {
             IsConnected = false;
         }

      void pulse_Oximeter_ProbeAttached(PulseOximeter
      sender)
         {
             IsConnected = true;
         }

      void pulse_Oximeter_Heartbeat(PulseOximeter sender,
             PulseOximeter.Reading reading)
         {
             MeasuredPulse = reading.PulseRate.ToString
             ("F1");
         }

  </Pulse_oximeter>
   ....
   ....
   ....
</platform>
```

Figure C.1. Code template fragment presenting action methods of the Pulse sensor

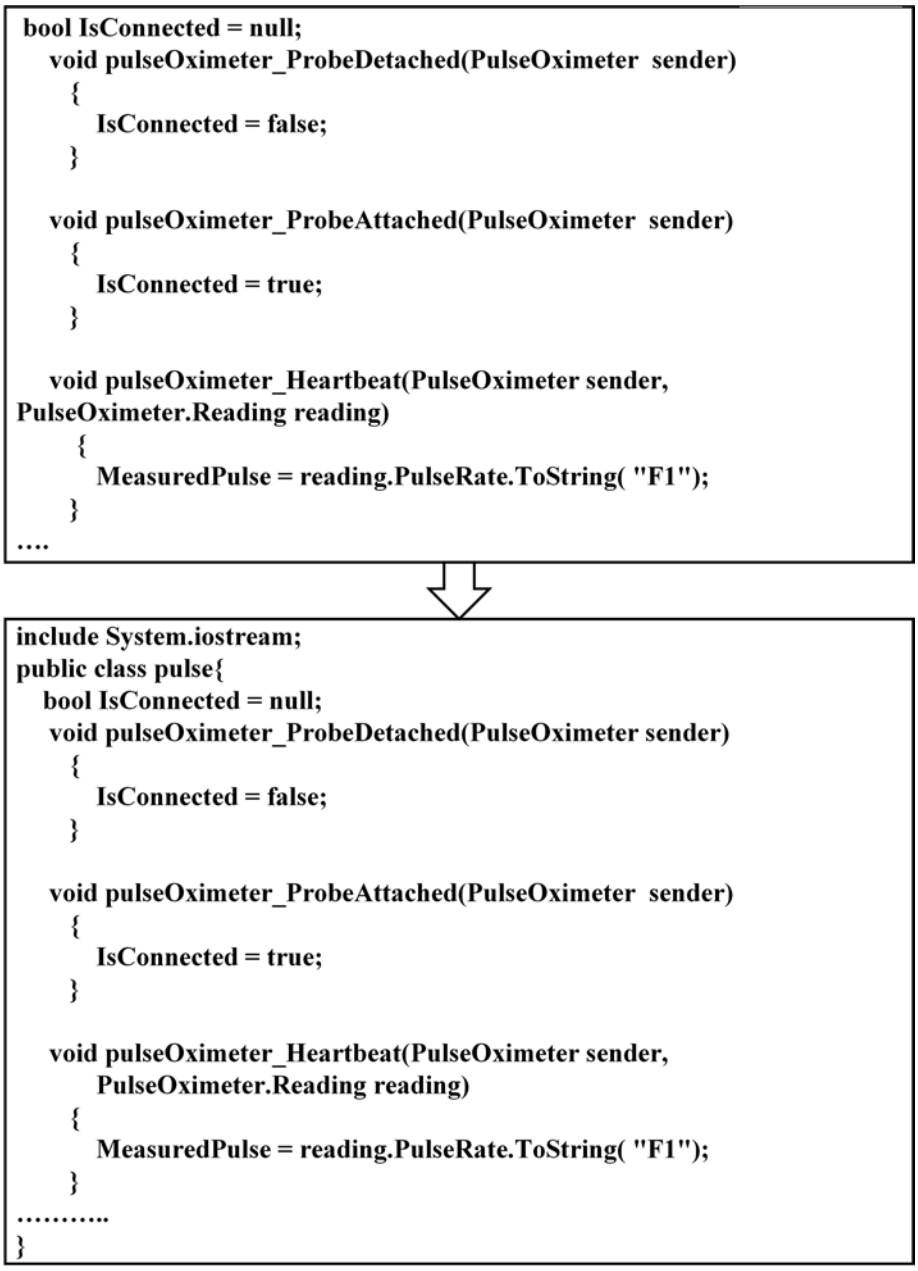


Figure C.2. Code class generation from the selected code fragment

```
Pulse:
void Interface_NetworkAddressChanged(object sender, EventArgs e)
 void NetworkChange_NetworkAvailabilityChanged(object sender, NetworkAvailabilityEventArgs e)
 void Interface_WirelessConnectivityChanged(object sender,
WiFiRS9110.WirelessConnectivityEventArgs e)
 void pulseOximeter_ProbeDetached(PulseOximeter sender)
 void pulseOximeter_ProbeAttached(PulseOximeter sender)
 void pulseOximeter_Heartbeat(PulseOximeter sender, PulseOximeter.Reading reading)
 void DataSending(GT.Timer timer)
  void DataReceiving()
```

```
Temperature:
void DataSending(GT.Timer timer)
 void bluetooth_BluetoothStateChanged(Bluetooth sender, Bluetooth.BluetoothState btState)
 void bluetooth_DeviceInquired(Bluetooth sender, string macAddress, string name)
 void bluetooth_connect()
 void temperatureHumidity_MeasurementComplete(TemperatureHumidity sender, double temperature,
double relativeHumidity)
```

```
Gas:
void DataSending(GT.Timer timer)
void GasSence_Preparation()
void zigbee_Initialization()
```

Figure C.3. Selected methods from the code template library