ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 1, No. 2, 2016

COMPUTER DEVICES AUTOMATIC SYNTHESIS AS A SERVICE FOR FPGA-BASED SMART-SENSORS OF CYBER-PHYSICAL SYSTEMS

Viktor Melnyk, Ivan Lopit, Andrii Kit

Lviv Polytechnic National University, 12, Bandera str., Lviv, 79013, Ukraine
Authors e-mail: viktor.a.melnyk@gmail.com

Submitted on 19.12.2016

Abstract: **Present paper is dedicated to the problems of studying and developing the theoretical and methodological framework, algorithmic base and corresponding software means to organize and realize the automatic synthesis of computer devices in the reconfigurable hardware platforms of the smart-sensors in cyber-physical systems with no human assistance. To solve this task, the following basic approaches will be used: a) a method of self-configuring of the computer system with reconfigurable logic; b) a "Software as a Service" software delivery model via a computer network; and c) an "Internet of Things" technology. The method of computer devices automatic synthesis in the reconfigurable hardware platforms of the smart sensors of the cyber-physical systems will be proposed. The client-server protocol of information exchange between the reconfigurable hardware platforms of the cyber-physical system measuring and computing nodes will be developed for automatic creation of computer devices in them. On the basis of the above protocol, the technical requirements to realization will be formulated and the principles of design and the main algorithms of the software interface operation will be developed. The program interfaces of realizing the protocol of information exchange between the reconfigurable hardware platforms of the smart-sensors for automatic creation of computer devices will be modeled and the results of their implementation and testing will be demonstrated.**

Index Terms: **Cyber-Physical System, Field Programmable Gate Array, FPGA-Based Smart-Sensor, Self-Configuring, Software as a Service, Internet of Things.**

I. INTRODUCTION

Over several years, developing and studying the cyber-physical systems (CPSs) continues to be one of the principal scientific and technical trends of the computer, information-communication and information-measuring systems progress. The cyber-physical system means a combination of physical processes and cybernetic means to ensure organizing the measuring and computing processes, protected storing and exchanging the measuring and service information, making decisions and organizing and realizing the influence on the physical processes. Combining these components within the framework of a single system allows the new results to be obtained capable of being used in creating a wide range of fundamentally new scientific, technical and service tools [1].

Due to automation of the information-measuring and computing processes, decision making processes and influencing the physical processes, CPSs are reasonably considered as one of the factors of the fourth industrial revolution. In 2012, the scientific research in the CPS sphere was recognized as one of the key trends of those carried out by the US National Science Foundation.

The further CPS development depends essentially on the technological progress in computer engineering and information-communication technologies. An important issue here is design of the energy saving autonomous measuring and computing nodes capable of not only the information collecting and primary processing but also of performing specialized computations. Equally important is also the increase of productivity and 'intellectualization' of computational means that must be able to analyze the huge data arrays in real time and to make necessary decisions.

Specialization and hardware interpretation of algorithms to be executed are the basic approaches in providing high productivity and energy efficiency of computer means. However, the use of the above approaches ensures high productivity indices of computer means in the related classes of problems only. Constructing computer means using reconfigurable components allows the problem of the efficient combination of the executed algorithm flexibility and hardware interpretation to be solved. The structure and functions of the above means might be readjusted in a prescribed manner with the purpose to take into account the structural and computational characteristics of the algorithms to be executed. Practical implementation of the reconfigurable computing technology has become possible since the advent of the field programmable gate array (FPGA) electronic circuits of a high integration degree.

Due to the aforementioned properties, FPGAs are perfectly suited for use in CPSs, in particular, to realize the measuring and computing nodes. One of such application is, for instance, protection of information transmitted between these nodes, since cryptographic algorithms used in the wireless networks with autonomous nodes, in particular, in the sensor [2] and Wi-Fi [3] ones are complicated, and their execution requires significant energy resources. Therefore, the information protection subsystems in such nodes are often realized in FPGAs. Realization of specialized computations within the measuring and computing node is the other example of FPGAs application, because this ensures low energy consumption at acceptable productivity.

II. RELATED WORK

In Ref. [1], a generalized structure of the cyber-physical system was proposed. It includes such components as high-performance computational means, information collection and processing centers, information protection and access management systems, communication environment, and an *N* autonomous measuring and computing nodes each interacting with the physical environment via the sensor and executive system. The measuring and computing node functions and the order of the node interaction with the physical environment are generally determined by the CPS purpose, architecture and functions as well as by the node location in it, and, if necessary could be modified. The node-embedded system of specialized computations and decision making is responsible for the node operation organization.

Due to the progress in the integrated circuits (ICs) and sensors production technologies, today the measuring and computing nodes are realized frequently in the integrated form. Portable devices that combine sensor system, computational means, communication means and a set of means for executing “intellectual” functions, i.e. decision making, self-diagnostics, self-testing etc., are called the smart sensors [4]. The latter involve usually a sensor itself, a microprocessor, a memory and an interface controller(s).

Minimal energy consumption is one of the principal requirements to the smart sensors. This is due to the fact that the measuring and computing nodes in CPS are autonomous, often have no fixed power source and are powered by the replaceable batteries. As one of consequences, the task of realizing the energy-efficient hardware means for specialized computations and decision making characterized by a low power consumption does arise.

Obviously, the use of high-performance universal processors like *Xeon*, *Pentium* or *Core i7* produced by *Intel*, or *Sempron*, *Athlon*, *A8* produced by *AMD* should not be reasonable here. On the other hand, computing capabilities of low-powered energy saving processors, e.g. *G-Series* produced by *AMD*, or *ARM Cortex*, are not always sufficient for many CPS applications. Therefore, the use of FPGAs is absolutely feasible since they contain the reconfigurable logic arrays, arithmetic devices, multipliers and even digital signal processor units for specialized computations, embedded memory for data and program storage, standard interface controllers. At the same time they are characterized by high potential productivity and relatively low power consumption. In addition, many modern FPGAs contain universal programmable microprocessors that could be used for specialized computation management and decision making. These microprocessors could be integrated into the FPGA chips as full custom regions or delivered in a form of soft cores and synthesized in FPGA together with other specialized computational means. The full-custom microprocessors implementations are, for example, *PowerPC™ 405* in the *Virtex-4* FPGA produced by *Xilinx* and *ARM922T™* in the *Excalibur* FPGA produced by *Altera*. The most widely used microprocessor soft cores are the *VHDL*-models of the 32-bit *LEON3* processor with the *SPARC V8* architecture produced by *Aeroflex Gaisler*, the 32-bit embedded *Nios* II processor produced by *Altera* with the architecture adapted to their own FPGAs, as well as *MicroBlaze*, – the 32-bit embedded *RISC*-processor produced by *Xilinx* with the Harvard architecture adapted to their own FPGAs.

The FPGA-based smart sensors are quite widespread today. Of their applications, one may distinguish the monitoring systems [5], computer vision and digital signal processing systems [4] and metrological systems [6]. The FPGAs in smart sensors are the reconfigurable hardware platforms (RHPs) intended to realize the specialized computational and decision making means. They are used to realize the application-specific processors for execution algorithms of, for example, compression [7]-[9], sound processing [10], [11], image processing [12]-[14], cryptographic transformations [15], [16]. The areas of the smart sensors application are analyzed in Ref. [14]. Of particular interest is using in the smart sensors those FPGAs that support partial reconfiguration. This enables, on the one hand, to extend their functions, when different application-specific processors operate in the different reconfigurable regions of FPGA and, on the other hand, to optimize energy consumption by adjusting the application-specific processors in FPGA to the data processing in real time at minimal frequency [13].

III. PROBLEM STATEMENT

A problematic issue of the FPGA-based smart sensors use in CPSs is configuration, i.e. realization of processors for specialized computations, interface controllers and other components of the specialized computing and decision making system.

The problem is, first of all, that implementing into FPGA requires application-specific processors soft cores designing (or getting ready-made solutions from the third parties). Designing process is described in detail in Refs. [18], [19]. This process is rather laborious and requires significant financial and time costs because it involves architectural designing of the application-specific processors, soft cores development and debugging using the hardware description language, and their logic synthesis in the target FPGA. As a result of the logic synthesis, the FPGA configuration code is obtained. Note that it is necessary first to define and develop the algorithms to be executed by the specialized computing and decision making system.

Second, after the application-specific processor designing, a problem arises of how to load the configuration code into the reconfigurable hardware platform of the smart sensor. To do this it is necessary to connect the above sensor to computer with the relevant software being installed and to execute the configuration procedure. Taking into account the scale of the CPS that may involve hundreds or thousands geographically distant measuring and computing nodes, one may realize how this process is longstanding and complicated. Moreover, changing functions or even adjusting operating parameters of the measuring and computing node may require FPGA reconfiguration, and this makes CPS rigid, inert and hardly suitable for modernization or reprofiling.

Therefore, the important task is developing the theoretical and methodological framework, algorithmic base and corresponding software means to organize and execute the automatic synthesis of computer devices in the reconfigurable hardware platforms of the smart sensors (and other types of measuring and computing nodes) of CPSs with no human assistance. In our opinion, to solve this task, the basic are applications of following methodological and technological approaches: a) a method of self-configuring of the computer system with reconfigurable logic; b) a “Software as a Service” (SaaS) software delivery model via a computer network; and c) an “Internet of Things” (IoT) technology. The use of the first approach allows the process of developing the application-specific processors soft cores to be synthesized in the reconfigurable hardware platforms of the smart sensors to be automated. The second approach enables this process to be realized as the service for the CPS smart sensors. Third approach aims at ensuring initiating the RHP configuration creation and its transfer to the smart sensor with no human assistance. Solving this task is a subject of research presented in this paper.

IV. STRUCTURE OF THE ARTICLE

The material of the present paper is structured as follows.

Section V reviews the basic methodological and technological approaches to the solution of the problems of the FPGA-based smart sensors application in the CPSs, namely, the method of self-configuring of the computer system with reconfigurable logic, the SaaS model and the IoT technology.

The method of computer device automatic synthesis in the reconfigurable hardware platforms of the CPS smart sensors is proposed in Section VI, making a basis for the development of the relevant algorithmic base and software means.

Section VII is devoted to the development of the client-server information exchange protocol with the reconfigurable hardware platforms of the CPS smart sensors for automatic creation of computer devices in them. The types of messages of client-to-server exchange and their transfer environment are reviewed; the operating algorithms and the relevant server’s and client’s finite state machines are developed. The example of a client-to-server communication is given.

In order to provide communication according to the above protocol, the data packet format is developed in Section VIII.

In Section IX, the requirements to the program interface that realizes the developed protocol of information exchange between the reconfigurable hardware platforms of the CPS smart-sensors for automatic creation of computer devices in them are determined, the software means of realizing this protocol are modeled and the results of their implementation and testing are demonstrated.

V. REVIEW OF BASIC APPROACHES TO SOLVING THE PROBLEMS OF THE FPGA-BASED SMART SENSORS APPLICATION IN CPSS

A. Method of Self-Configuring of the Computer System with Reconfigurable Logic

The self-configurable computer system (SCCS) is the computer system with reconfigurable logic where the program compilation includes automatically performed actions of creation of configuration, and which acquires that configuration automatically in the time of program loading for execution [18], [20], [21].

The SCCS automatically executes: 1) computational load balancing between the general-purpose processor and reconfigurable environment (RCE); and 2) creation of an application-specific processor (ASP) HDL-model. Loading of the configuration files obtained after logical synthesis into the RCE is carried out by the operating system in parallel with loading of the general-purpose processor’s subprogram executable file into the main memory after program initialization [18], [20], [21].

The method of information processing in the SCCS consists of three stages: compiling the program, its loading, and execution. This method supposes following. The user creates a program written in a high-level programming language and submits it into the SCCS. During compiling the SCCS automatically performs the following actions: divides this program into the general-purpose processor’s subprogram and RCE’s subprogram, performs general-purpose processor’s subprogram compilation and generates its executable file, creates ASP’s HDL-model to perform RCE’s subprogram, performs ASP’s logic synthesis, and stores obtained executable and configuration files into the secondary storage.

At the stage of the program loading after its initialization, the SCCS loads the executable file of the general-purpose processor’s subprogram into the main memory using a conventional loader and, at the same time, loads the configuration files into the RCE and thus creates an ASP in there using the FPGA configuring tools. Then, the stage of the program execution is performed.

The use of the method of self-configuring in the FPGA-based smart sensors allows one to generate automatically the computer devices soft cores to be synthesized in their RHPs, to obtain their configurations and, thus, to solve the problem of developing the application-specific processors, interface controllers and other components of the specialized computing and decision making system. At the same time, the complexity of the method of self-configuring and the need in the high-performance computer means for its execution give no possibility to realize it inside the smart sensor with its limited energy and computational resources.

B. "Software as a Service" Software Delivery Model

Software as a service (SaaS) is a service delivery model in which software is centrally hosted and is accessible for clients [22], [23]. SaaS has become a common delivery model for many business applications, including software for office and messaging, database management, computer-aided design, gaming, antivirus protection and more [24].

The vast majority of SaaS solutions are based on a multi-tenant architecture. With this model, a single version of the application, with a single configuration (hardware, network, operating system), is used for all customers ("tenants"). To support scalability, the application is installed on multiple machines. This is contrasted with traditional software, where multiple physical copies of the software – each potentially of a different version, with a potentially different configuration, and often customized – are installed across various customer machines.

The use of the SaaS model in the CPS allows one to organize via the network the provision to the smart sensors of the software tools necessary for the automatic generation of the computer device soft cores to be synthesized in their FPGAs and to obtain their configurations. Obviously, these software tools must be installed at the special server in the CPS.

C. "Internet of Things" Technology

The Internet of things (IoT) is the information communication technology allowing the interaction of physical devices, vehicles, buildings, and other items – equipped with embedded electronic hardware, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data [25], [26]. The IoT technology allows objects to be sensed and/or controlled remotely across existing network infrastructure, creating opportunities for direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit. The IoT is one of the basic technologies used in the more general class of cyber-physical systems. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure.

"Things", in the IoT sense, can refer to a wide variety of devices. Typically, IoT is expected to offer advanced connectivity of devices, systems, and services that goes beyond machine-to-machine (M2M) communications and covers a variety of protocols, domains, and applications.

We suggest using the CPS smart sensors as the "things" interacting via the IoT with the goal to automate the configurations creation requests and receipt. This will enable the configuration codes to be loaded automatically into the smart sensor's RHP in case of a need in the change of operating algorithms of the specialized computing and decision making system and will solve a problem of the CPS rigidity and inertia during its reprofiling or modernization.

The benefits that may be gained due to the use of the above basic approaches in the CPSs with the FPGA-based smart sensors are generalized in Fig. 1 in a form of a "*Self-Configurability-SaaS-IoT*" triangle. Combining these benefits forms an approved theoretical basis for the development of the method of computer devices automatic synthesis in the RHPs of the CPS smart sensors, which is presented below.

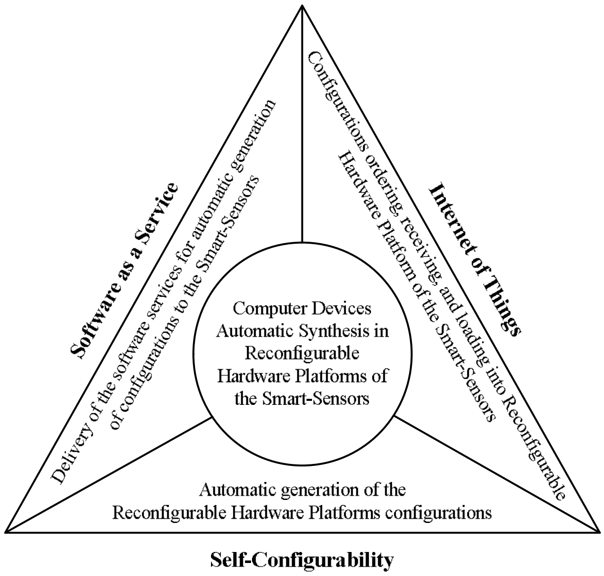

Software as a Service
Delivery of the software services for automatic generation of configurations to the Smart-Sensors
Internet of Things
Configurations ordering, receiving, and loading into Reconfigurable Hardware Platform of the Smart-Sensors
Computer Devices Automatic Synthesis in Reconfigurable Hardware Platforms of the Smart-Sensors
Automatic generation of the Reconfigurable Hardware Platforms configurations
Self-Configurability

Fig. 1. "Self-Configurability-SaaS-IoT" triangle

VI. METHOD OF COMPUTER DEVICES AUTOMATIC SYNTHESIS IN THE RECONFIGURABLE HARDWARE PLATFORMS OF THE CPS SMART SENSORS

As stated above, today the measuring and computing nodes are being realized mainly as the integrated devices called the smart sensors. Therefore, when describing this method below, we shall use just this term, not limiting, however, its applications to the other types of the measuring and computing nodes.

To realize the computer device automatic synthesis in the reconfigurable hardware platforms of the smart sensors, one has to introduce into the CPS structure [1] additionally the following components:

– a library to store the high-level descriptions of the operating algorithms of the specialized computing means. Such library could be both global, i.e. accessible for many smart sensors, and local one. The high-level descriptions are presented in a form of the computer programs written in the programming language, for instance *C*. They may be put into the library during the CPS creation or later, during its reprofiling or modernization;

– a complex of software tools for automatic generation of smart sensors configurations (hereinafter referred to as the configurations generation system). This system may be involved in the CPS high-performance computational means or be its separate component connected to the smart sensors via the communication environment.

Fig. 2 illustrates the method of computer devices automatic synthesis in the RHPs of the CPS smart sensors. The essence of this method lies in that:

– the smart sensor receives from the library the P_{HLL} program that describes the operation algorithm of the specialized computing means in it and transfers this programs together with the universal microprocessor (*MPC*) and RHP (*RHPC*) characteristics to the configurations generation system;

the configurations generation system, having received the above program and characteristics, shall automatically execute the following actions:

– extracts from the P_{HLL} program most computationally complex fragments and distributes it onto the P_{MP} subprogram of the universal microprocessor and the RHP subprogram P_{RHP} constructed from the extracted fragments;

– compiles the P_{MP} subprogram of the universal microprocessor into the executable file *obj* that corresponds to its architecture and the RHP subprogram to the configuration file *conf* that corresponds to its characteristics. In this case the RHP subprogram compilation into the configuration file consists in the automatic creation of the application-specific processor soft core from this subprogram with further logic synthesis execution;

– transfers the universal microprocessor executable file created and the RHP configuration file to the smart sensor;

– the smart sensor receives the above files, stores the executable file into the memory and loads the configuration into the RHP.

In the method described, the smart sensor is a client, whereas the configurations generation system is a server.

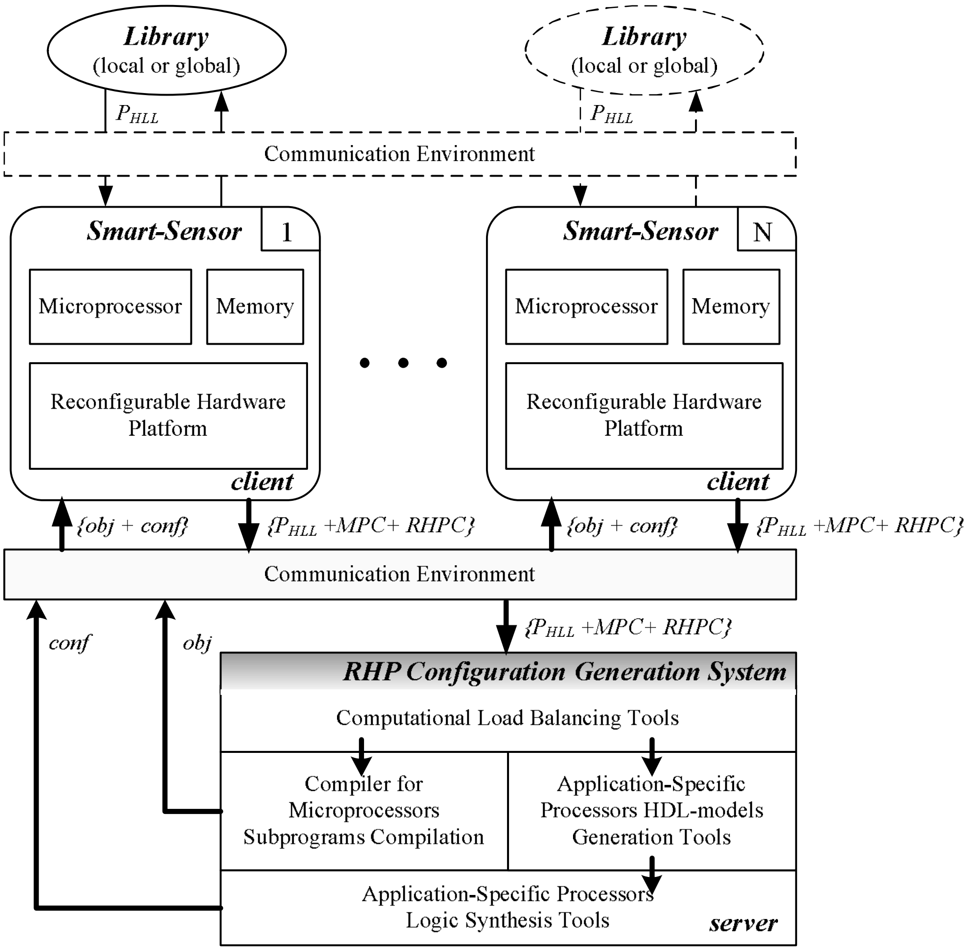

Library
(local or global)
Library
(local or global)
P_{HLL}
P_{HLL}
Communication Environment
Smart-Sensor
1
Microprocessor
Memory
Reconfigurable Hardware Platform
client
Smart-Sensor
N
Microprocessor
Memory
Reconfigurable Hardware Platform
client
{obj + conf}
{P_{HLL} +MPC+ RHPC}
{obj + conf}
{P_{HLL} +MPC+ RHPC}
Communication Environment
conf
obj
{P_{HLL} +MPC+ RHPC}
RHP Configuration Generation System
Computational Load Balancing Tools
Compiler for Microprocessors Subprograms Compilation
Application-Specific Processors HDL-models Generation Tools
Application-Specific Processors Logic Synthesis Tools
server

Fig. 2. Method of computer devices automatic synthesis in the RHPs of the CPS smart sensors

To carry out the above actions, one has to introduce into the configurations generation system the following software tools:

1. Tools for computational load distribution between the universal processor and RPH. These tools find automatically in the P_{HLL} programs those fragments that constitute the computational load and provide at the execution in RPH the advantage in the smart sensor energy consumption, and distribute the P_{HLL} program onto the P_{MP} subprogram of the universal microprocessor replacing in it the extracted fragments by the RPH instructions, and the RPH subprogram P_{RHP} formed of the above fragments. One of the examples of such system realization is given in Ref. [27].

2. Compiler(s) for the P_{MP} subprograms compilation from the input language, they are written in, into the object codes obj that may be directly executed by the universal microprocessor of the smart sensor.

3. Computer device soft cores generation tools that generate automatically the application-specific processors soft cores from the P_{RHP} subprograms, for example, *Chameleon* produced by *Intron* [28], [29], *Agility Compiler* [30] and *DK4 Design Suite* [31] produced by *Celoxica*, *CoDeveloper* produced by *Impulse* [32].

4. Logic synthesis tools for application-specific processors soft cores synthesis in target FPGAs. These tools are available from the FPGA vendors, e.g. *Vivado Design Suite*, *ISE*, *Alliance*, *Foundation* produced by *Xilinx* [33]; *Quartus II*, *Max + II* produced by *Altera* [34].

As mentioned above, the smart sensor transmits to the configurations generation system the universal microprocessor and RPH characteristics together with the P_{HLL} program. Let's consider the purpose of these characteristics.

The universal microprocessor architecture is its characteristic that makes a basis for selecting the compiler from the input language to the object code.

The following RHP characteristics are classified as the basic ones: the FPGA type and series, the package type and the number of pins. The secondary characteristics are as follows: the embedded memory units amount and organization, the arithmetic and logic devices amount and organization, the basic logic elements and the input/output blocks amount, the maximal operating clock frequency and the energy consumption. The basic RPH characteristics are the input information for the logic synthesis tools. The secondary RHP characteristics are necessary for the computer device soft cores generation tools to determine the application-specific processors parameters, in particular, the number of the parallel computational units, the maximal command memory size, the interface capacity etc.

Thus, the task of the RHP configurations automatic creation and receipt in the measuring and computing nodes with no human assistance is solved in CPS.

Having the above software tools in hand, it is necessary to develop the protocol of information exchange in CPS between the configurations generation system and the smart sensors for automatic synthesizing the computer devices in their RHPs, as well as the data packet format for this information transfer. We shall examine these problems below.

VII. PROTOCOL OF INFORMATION EXCHANGE IN CPS BETWEEN THE CONFIGURATIONS GENERATION SYSTEM AND SMART SENSORS FOR COMPUTER DEVICES AUTOMATIC SYNTHESIS IN RHP

A. Messages and Their Communication Environment

Communication between the server, i.e. the configurations generation system, and the client, i.e. the smart sensor, is realized in a way of transferring messages. The communication environment must ensure the transfer possibility, while the transfer protocol must warrant the message delivery, as, for instance, *TCP* [35] or other data transfer protocols with guaranteed delivery in the *OSI* model [36].

The messages in the protocol could be divided into 2 groups:

A group of messages for connection state management. This group includes those messages that are responsible for connection establishment, authorization, its passing, control and completion.

A group of messages for the data transfer. This group includes those messages that contain information on the P_{HLL} programs, characteristics of the microprocessor and RHP and the RHP configuration itself.

B. Server's Finite State Machine

Consider first a server's finite state machine responsible for connection state management from the server's side (Fig. 3). The state machine operates according to the following algorithm.

The initial state of the state machine, which the server starts its operation from, is *Init*. After calling *start*(), the server transits into the *WaitForConnection* state, where it waits for a client connection.

If the connection is successful, the *DoAcceptConnection*() method is called, and the server transits into the *Connected* state.

If the connection error occurs or the number of connection attempts is exceeded, the server transits into the *ConnectionFailed* state by means of the *OnMaxConnectionAttempt*() and *OnConnectionFailed*() methods, respectively.

If the number of connection attempts is not exceeded, the transition into the *WaitForConnection* state occurs by means of the *WaitForNewConnection*() method. Otherwise, the transition into the *Error* state will take place by means of the *HandleConnectionFailed*() method that is responsible for irreversible connection error.

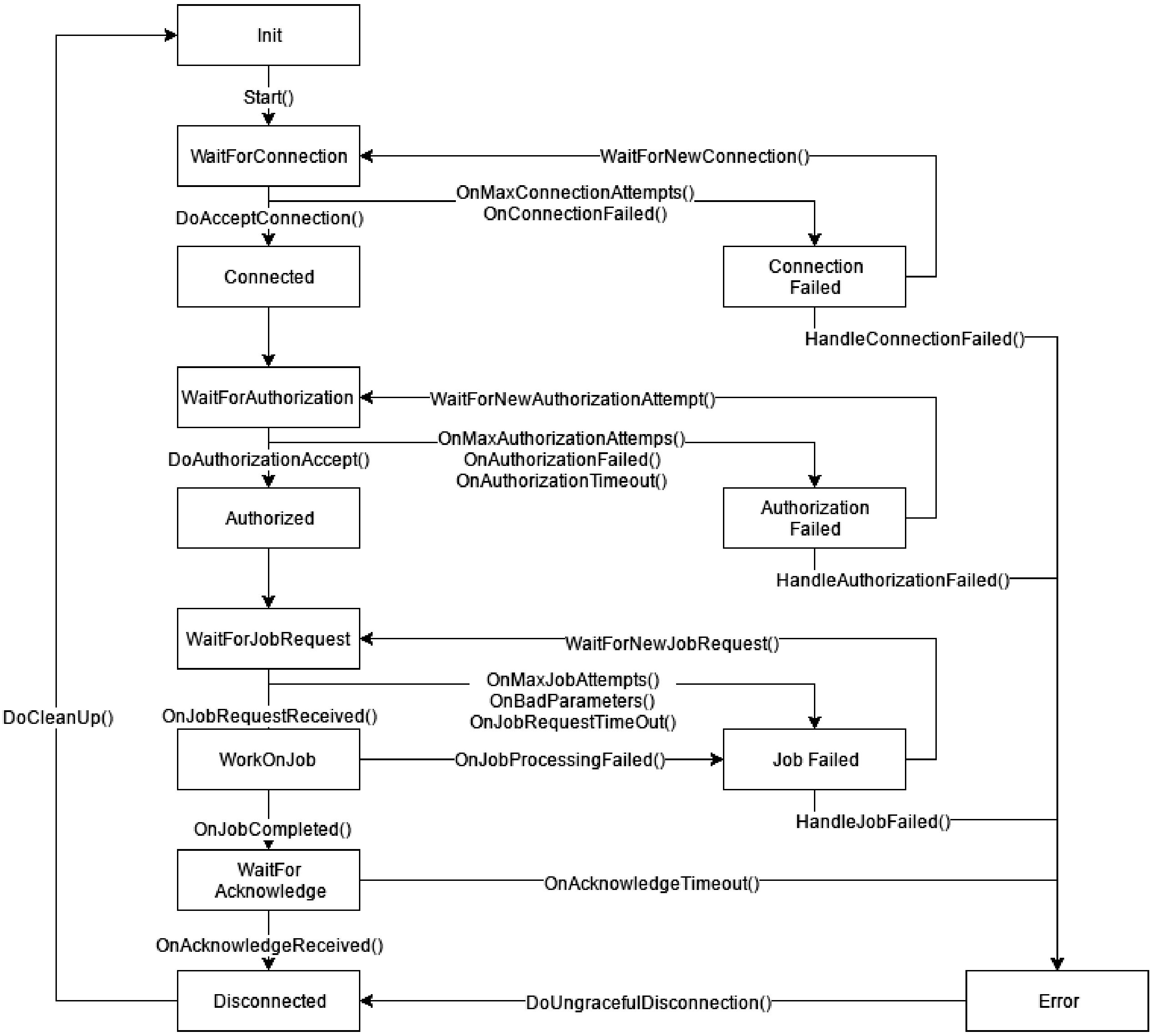


Fig. 3. Server's finite state machine

At the successful connection, the server will transit to the *WaitForAuthorization* state, i.e. into the machine state, in which the server waits for the client authorization.

If the client is authorized, the server transits into the *Authorized* state by means of *DoAuthorizationAccept*(). In case of the authorization error, the *OnAuthorizationFailed*() method is called and the server transits into the *AuthorizationFailed* state.

If the number of authorization attempts is not exceeded, the server may return from the *AuthorizationFailed* state into the *WaitForAuthorization* state by calling the *WaitForNewAuthorizationAttempt*() method. Otherwise, the *HandleAuthorizationFailed*() method will be executed and the server will transit into the *Error* state.

If the limit of the server stay in the *WaitForAuthorization* state is exceeded, the *OnAuthorizationTimeout*() method is called and the server transits into the *AuthorizationFailed*() state.

In case of a successful authorization, the server will transit into the *WaitForJobRequest* state. In this state, the server waits for the request for the RHP configuration generation execution. If the limit of the server stay in the *WaitForJobRequest* state is exceeded, the *OnJobRequestTimeOut*() method will be called and the transition into the *JobFailed* state will occur. In case the bad synthesis parameters were transmitted, the transition to the *JobFailed* state will occur by means of *OnBadParameters*(). The transition from the *JobFailed* state into the *WaitForJobRequest* one is possible by means of *WaitForNewJobRequest*(), provided the number of execution requests is not exceeded, otherwise the transition to the *Error* state occurs and connection will be finished. If *OnJobRequestReceived*() is received successfully, the server will transit into the *WorkOnJob* state, where configurations for RPH are generated. Errors may occur during execution, and in this case the server will transit from the *WorkOnJob* state into the

JobFailed one. As a result of successful synthesis, the *OnJobCompleted*() method will be executed successfully, and the server will transit into the *WaitForAcknowledge* state, where it will wait for the client's response that acknowledges the configuration receipt. In case this does not occur, the server will transit into the *Error* state by means of the *OnAcknowledgeTimeout*() method. If the acknowledgement is received, *OnAcknowledgeReceived*() is called. The server will transit into the *Disconnected*() state, where connection terminates. After the connection termination, the state is reset by the *DoCleanUp*() method and the server transits into the *Init* state. In case of the irreversible error, the server will stay in the *Error* state. The transition from this state into the *Disconnected* one will occur by means of the *DoUngracefulDisconnection*() method.

C. Client's Finite State Machine

This finite state machine is responsible for the connection management from the client's side (Fig. 4).

The client's finite state machine operates according to the following algorithm. The client starts operating from the *Init* state. Connection with the server is executed by means of the *DoConnect*() method and the client transits into the *Connecting* state waiting for the server response. If the response waiting time is exceeded, the *OnConnectionTimeOut*() method is called, and the client transits to the *ConnectionFailed* state. In the case when the server fails to accept connection, the *OnHostRejects*() method will be called, and the client will transit into the *ConnectionFailed* state.

The transition from the *ConnectionFailed* state to the *Connecting* state is possible by means of the *DoConnectionAttempt*() method provided if the number of attempts did not exceed the limit. If the connection is successful, the client transits into the *Connected* state. By calling the *DoAuthorization*() method, the transition is performed from the *Connected* state to the *Authorizing* one, in which the client waits for the authorization from the server.

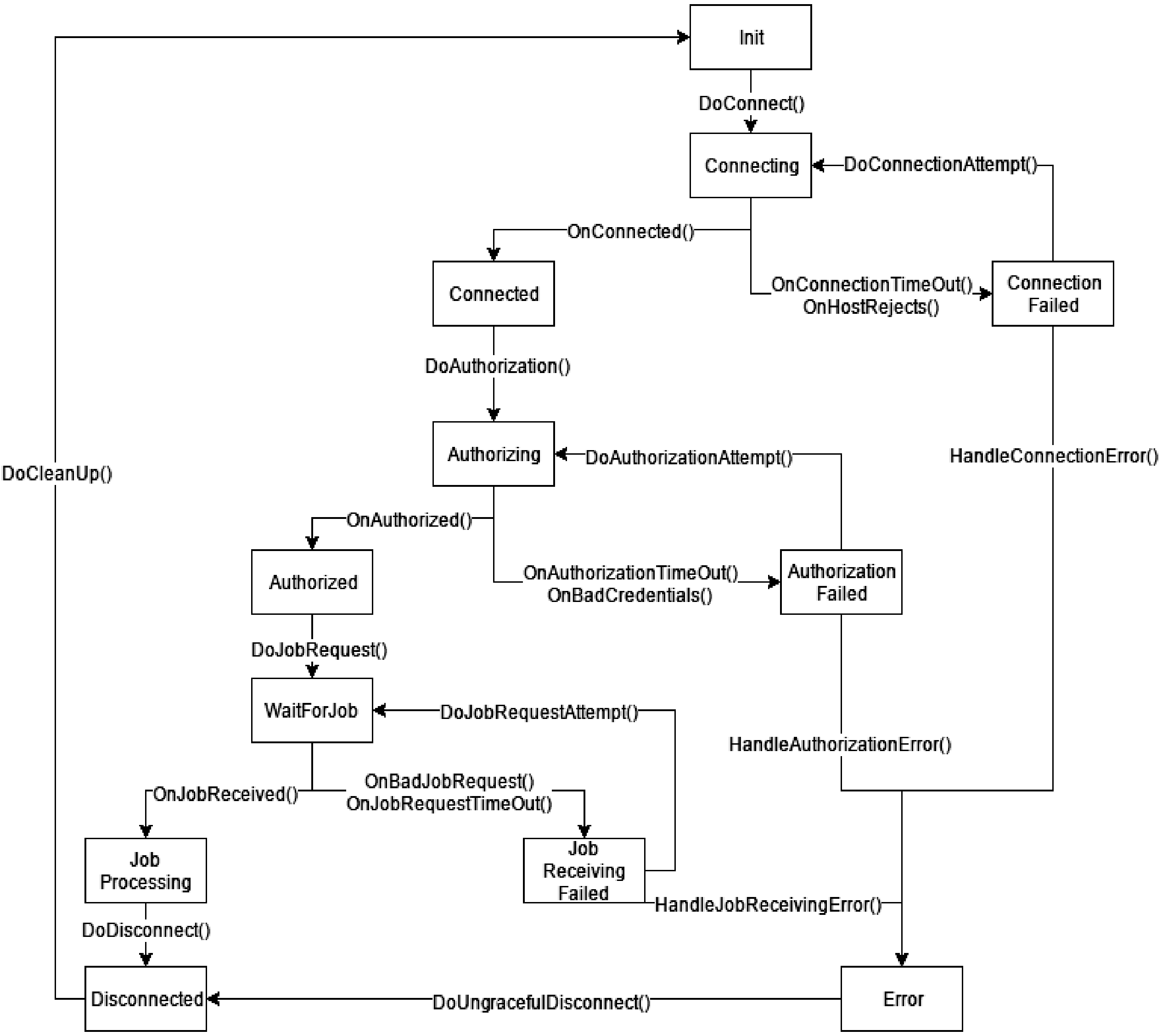


Fig. 4. Client's finite state machine

If the authorization is not successful or the response waiting time is exceeded, the transition into the *AuthorizationFailed* state occurs by means of the *OnAuthorizationTimeOut*() or *OnBadCredentials*() methods.

If the limit of attempts is not exceeded, the *DoAuthorizationAttempt*() method is executed, otherwise the transition to the *Error* state occurs by means of the *HandleAuthorizationError*() method.

If the authorization is successful, the *OnAuthorized*() call takes place and the server transits to the *Authorized* state. The client will transit from this state into the *WaitForJob* state by means of the *DoJobRequest*() call and will wait for the result of the RHP configuration generation. If during the request the errors occur, then, using the *OnBadJobRequest*() and *OnJobRequestTimeOut*() methods, the transition into the *JobReceivingFailed* state is executed. If the request limit is not exceeded, the transition into the *WaitForJob* state takes place by means of *DoJobRequestAttempt*().

After finishing the configuration generation, the *OnJobReceived*() method will be called and the client will transit to the *JobProcessing* state. The configuration will be stored in this state. By calling the *DoDisconnect*() method, the client will transit into the *Disconnected* state and then, having called *DoCleanUp*(), it will transit into the initial state *Init*. In case of the irreversible error, the server will stay in the *Error* state. The transition from this state into the *Disconnected* one occurs by means of the *DoUngracefulDisconnection*() method.

D. Example of the Client-to-Server Communication

Consider an example of the client's communication with the server (Fig. 5).

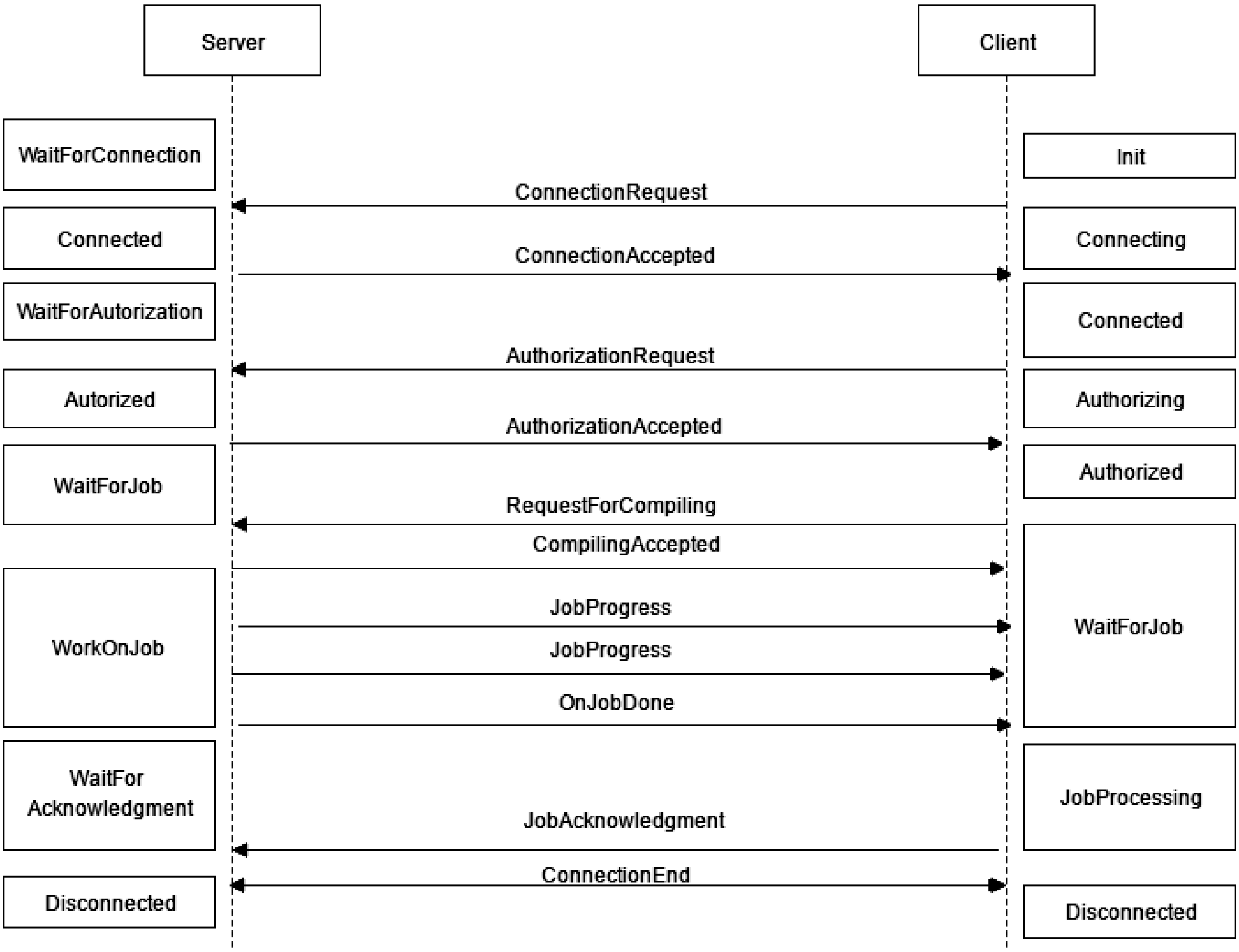
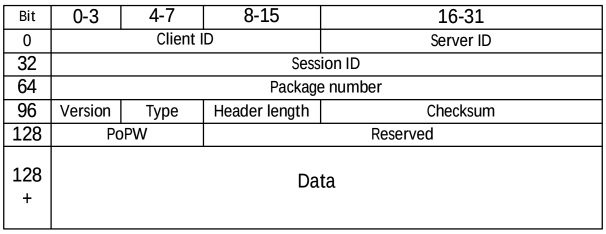


Fig. 5. Example of the client's communication with the server

When the server transits to the *WaitForConnection* state, the client will send the *ConnectionRequest* message and transit into the *Connecting* state, where it will wait for the server response. The server, in turn, receives this message, transits into the *Connected* state, sends *ConnectionAccept* and transits into the *WaitForAuthorization* state, where it will wait for authorization. The client, having received *ConnectionAccept*, transits into the *Connected* state, sends the *AuthorizationRequest* message and transits into the *Authorizing* state. After the receipt of *AuthorizationRequest*, the server transits into the *Authorized* state, sends the *AuthorizationAccepted* message and transits into the *WaitForJob* state, where it waits for the RHP configuration generation request. Having received *AuthorizationAccepted*, the client transits into the *Authorized* state, makes an RHP configuration generation request and then transits into the *WaitForJob* state. After the receipt of the configuration generation request (*RequestForCompilation)*, the server transits into the *WorkOnJob* state and, using the *CompilationAccepted* message, informs that the request is accepted. After completing the configuration generation, the server transits into the *WaitForAcknowledgment* state, where it waits for the client's response. Having got *OnJobDone*, the client processes the configuration received and sends the *JobAcknowledgment* message that acknowledges the successful configuration receipt and completes the communication session transiting into the *Disconnected* state. The server, having received the *JobAcknowledgment* message, terminates the communication session by transiting to the *Disconnected* state as well.

VIII. DATA PACKET FORMAT FOR INFORMATION EXCHANGE IN CPS BETWEEN THE CONFIGURATIONS GENERATION SYSTEM AND THE SMART SENSORS

Information exchange between the server, i.e. the configurations generation system, and the clients, i.e. the smart sensors, is performed in both directions. The client sends the program and the universal microprocessor and RHP characteristics, while the server sends the smart sensor configuration. This information makes the packet payload. To support connection and keep the management information, the packet must include the relevant header. Fig. 6 illustrates the packet structure, in which information is organized and the information exchange protocol operates.



Bit	0-3	4-7	8-15	16-31
0	Client ID			Server ID
32	Session ID			
64	Package number			
96	Version	Type	Header length	Checksum
128	PoPW		Reserved	
128 +	Data			

Fig. 6. Data packet structure for information exchange in CPS between the configurations generation system and the smart sensor

The packet format involves the following fields:

Client's and server identifiers. CPS may include a large number of smart sensors. The server that generates configurations could also be not one in the system. To identify unambiguously both client and server, one has to enter identifiers and indicate them in the header of the packets that the client and server exchange.

Session number. This field allows a particular streaming thread to be identified.

Packet serial number. Using this number, one may renumber packets in each particular session. In case of a loss of this number, a number that must be resent is indicated in the *serial number* field.

Protocol number. The current version of communication protocol for computer devices automatic synthesis in the RHPs of the CPS smart sensors is indicated in this field.

Packet type. Each packet type (i.e. connection request, connection acknowledgement, authentication, configuration transfer, packet resending request etc.) has a unique identifier assigned by the bit sequence. The information in the header fields and the packet data will be treated in a different manner depending on the packet identifier.

Header length. This field defines the packet header size in the 4-byte words.

Checksum. The hash value calculated for the entire packet is indicated in this field to control the transmitted information integrity.

Completion percentage. The percentage of information transferred from the server to the client and vice versa is indicated in this field.

Data. In this field, the program, RHP and the universal microprocessor characteristics, configuration, authentication data etc. are indicated depending on the packet type and transmission direction.

Reserved bits. Some bits of the packet are reserved for future needs to provide the protocol expandability.

IX. SOFTWARE MEANS FOR REALIZING THE PROTOCOL OF INFORMATION EXCHANGE IN CPS BETWEEN THE CONFIGURATIONS GENERATION SYSTEM AND THE SMART SENSORS

The problems of implementing the software means (SM) that realize the protocol developed are considered in this Section. The requirements to SM are defined, the basic modules and their components are described, and the relations between the modules and the module components are shown. To demonstrate the SM operability and to verify the protocol, the operation of one of the functional tests is exhibited.

A. Determining the Requirements to the Software Means of Protocol Realization

During the communication protocol realization one has to ensure its reliable operation and possibility to work with all necessary resources by providing it with appropriate flexibility. When forming the program interface requirements, the peculiarities of the CPS smart sensor architecture and the characteristics of the environment it will operate in must be taken into account.

The requirements to the software means of this protocol realization could be divided into two groups: the functional ones that describe what functions the SM must perform and the non-functional ones that describe how the above SM must operate and what should be their properties and characteristics [37]. The program interface must comply with the following functional requirements:

Realization of information exchange in CPS between the configurations generation system and the smart sensors for computer devices automatic synthesis in RHPs according to the protocol developed.

Flexibility. Since CPS can be oriented to a wide application range, the program interface should not depend on the particular application.

CPS dynamical structure support. The number and composition of the CPS objects that send and receive messages may vary dynamically during the CPS operation. In such a case, at the change of one object state, all other ones dependent of it will be informed about this event.

Data base operation support. There is a necessity to store and gain the access to the configuration files, object files, RHP characteristics and high-level descriptions of the operation algorithms of the specialized computing means.

The non-functional requirements define most frequently the qualitative characteristics of the SM under development. Consider the principal ones of them:

Productivity.

Accessibility – defines the time of the SM continuous operation.

Reliability – describes the SM behavior in the emergency situation (for instance: automatic restart, resumption of operation, data storage, important data duplication).

Data storage time.

Convenient use and support.

Data security.

Software means and entire system configurability.

Besides the aforementioned, the following non-functional requirements may be used at the development stage:

Possible component reuse.

Expandability and granularity. The program interface must consist of independent modules, each of them performing separate function. Such organization will give a possibility of its easy adapting to the changes in the CPS structure and to the appearance of the new functional requirements.

Scalability – a possibility of the horizontal and/or vertical scaling of system or components.

Encapsulation – hiding implementation details from the client. The clients will not require recompilation at the pointer type implementation change. If the dynamic library uses the opaque pointers, its modification will not result in the binary incompatibility with the applied programs [38].

Code portability and different hardware platforms support. The program interface must operate at different hardware platforms and must be compatible with the software to be used in different CPS realizations.

Compatibility with protocols that lay on the lower levels of the network model. The program interface must provide the information transfer possibility via Internet or local network, in particular, the wireless one.

Using of standard protocols and technologies of communication between the system components and with external software.

B. Realization of Some Non-Functional Requirements o SM at the Development Stage

The components reuse may be reached by representing them in a form of atomic components. If each component has only one purpose that complies with the "*single responsibility*" principle [39], then the complex components could be represented as the composition of the simpler ones.

Combining components, the authors used the *package principle* [40]. At that, the components that are varying or used simultaneously were combined into the modules (libraries). The authors also decided to put into the separate modules the file system utilities (*file_utils*), the strings and the *JSON* format utilities (*parsing_utils*), the part for work with network (*network*), and the inter-process communication module (*IPC*). The protocol realization itself is represented in the *ota* (*over the air*) module. The general structure of the SM modules of protocol realization and relations between them are shown in Fig. 7.

To ensure the SM code portability towards the other platforms, it is necessary to choose the cross-platform programming language. The authors settled on the *C++* language that belongs to the most common ones of this type, allows the advantages of the object-oriented programming to be used and supports simultaneously the low-level constructions of the *C*. To simplify compilation at different platforms, the *CMake* (*Cross-Platform Make*) [41] – a cross-platform system of the program code compilation automation, was used.

C. Communication Process Basic Components Modeling

The client, the server, the message, the access point to the communication environment and the communication environment itself are the basic components of the communication process. Below we will show an example of the sequence of operations that describes the communication between the above components:

at the one end, the client/server creates a message;

the client/server transmits the message to the access point;

the access point converts the message into the form convenient for transmission through the particular communication environment;

the message is transmitted through the communication environment;

at the other end, the access point receives the message and converts it into the form understandable for the server/client;

the access point notifies the server/client about the receipt of a new message.

The components must be independent and provide the possibility of substitution. For example, the access point construction depends entirely on the communication environment. If the TCP/IP protocol is used, it may be realized using *WinSockets* [42] or *Berkley Sockets* [43]. As an alternative, communication according to the *SOAP* protocol or in a form of the *HTTP* requests is possible. For the test needs, the communication environment could be also realized as a software in a form of a synchronous message queue.

Summarizing, it seems expedient to use the "*dependency injection*" principle [44] and the *Bridge* coding pattern [45] to realize the non-functional requirements. This will allow one to distinguish realization from abstraction and provide a possibility to use different communication environments with unchanged server-to-client communication logics. To provide the system with configurability, we shall use the *Abstract Factory* coding pattern [44].

The *UML*-notation of the 'message' component is shown in Fig. 8(a). Here *id* is a unique message identifier represented by the 64-bit unsigned integer; *type* is the message type represented by the enumeration type; *timestamp* is the time represented by the 64-bit unsigned integer; *body* is the message payload content in a form of a string.

To provide the flexibility of communication with the access point for the client and server the following interfaces are realized:

i_access_provider – the abstraction that allows the server or client to send messages via the access point;

i_access_listener – the abstraction that allows the server or client to receive notifications from the access point.

The *UML*-notations of these abstractions are shown in Fig. 8(b).

The access point is realized as the *class i_access_layer* abstraction, the *UML*-notation of which is presented in Fig. 8(c). Using the *register_access_listener* and the *unregister_access_listener* methods, the client and the server subscribe and unsubscribe from the access point notifications, respectively.

Combining the above communication process components, we create the *ota* module that implements the protocol. Its *UML* class diagram is presented in Fig. 9.

The components communication could be presented by the relevant *UML*-diagram (see Fig. 10).

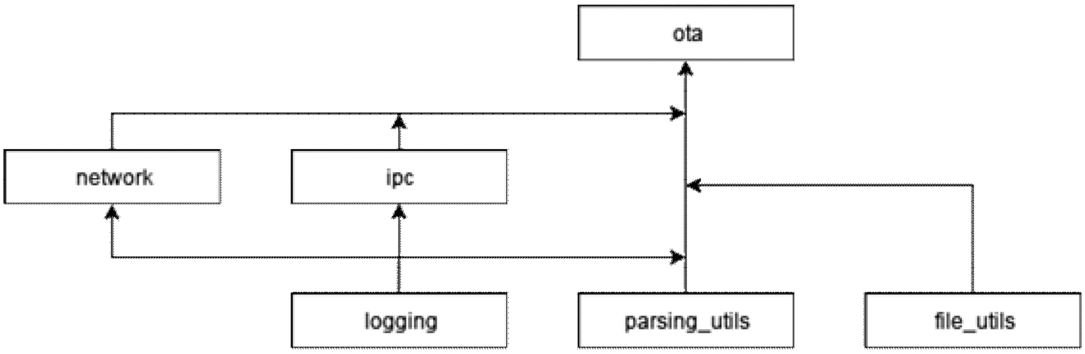


Fig. 7. General structure of the SM modules of protocol realization and relations between them.

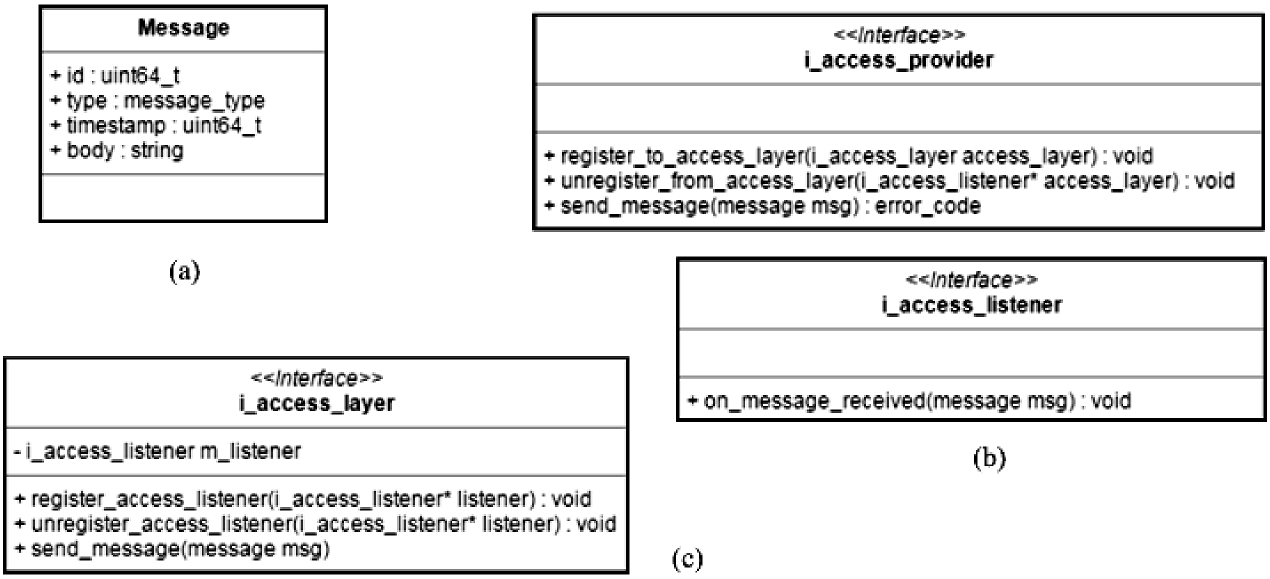

Message
+ id : uint64_t
+ type : message_type
+ timestamp : uint64_t
+ body : string
(a)
<<Interface>>
i_access_provider
+ register_to_access_layer(i_access_layer access_layer) : void
+ unregister_from_access_layer(i_access_listener* access_layer) : void
+ send_message(message msg) : error_code
<<Interface>>
i_access_listener
+ on_message_received(message msg) : void
(b)
<<Interface>>
i_access_layer
- i_access_listener m_listener
+ register_access_listener(i_access_listener* listener) : void
+ unregister_access_listener(i_access_listener* listener) : void
+ send_message(message msg)
(c)

Fig. 8. UML-notations of: (a) – 'message' component, (b) – i_access_listener and i_access_provider abstractions that realize the client's and the server interfaces, (c) – class i_access_layer abstraction that realizes the access point to the communication environment

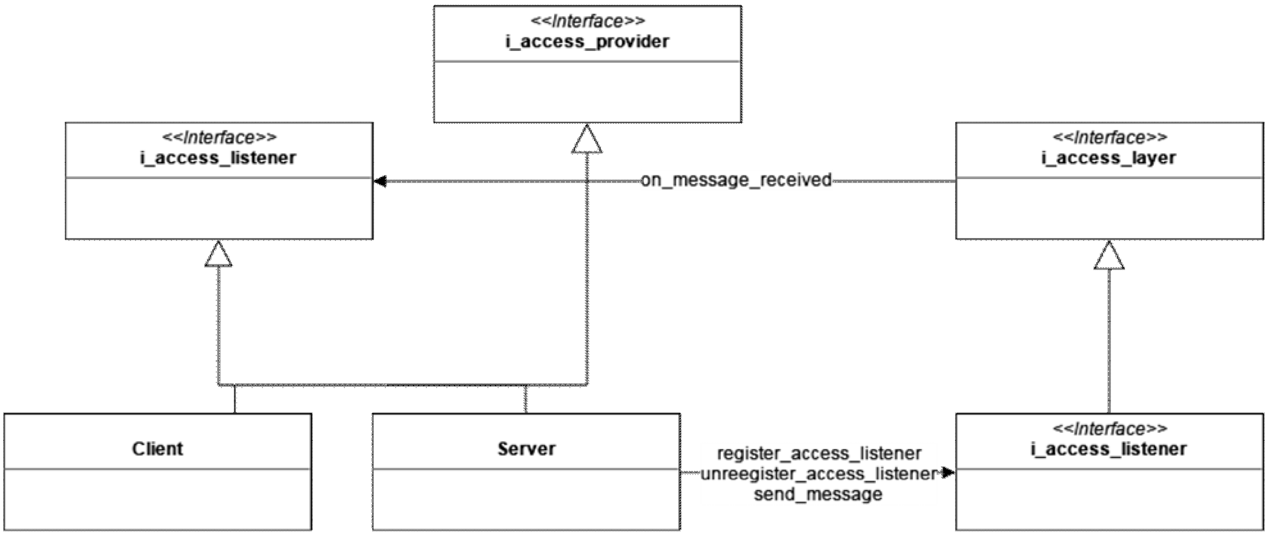


Fig. 9. UML class diagram of the ota module that implements the protocol.

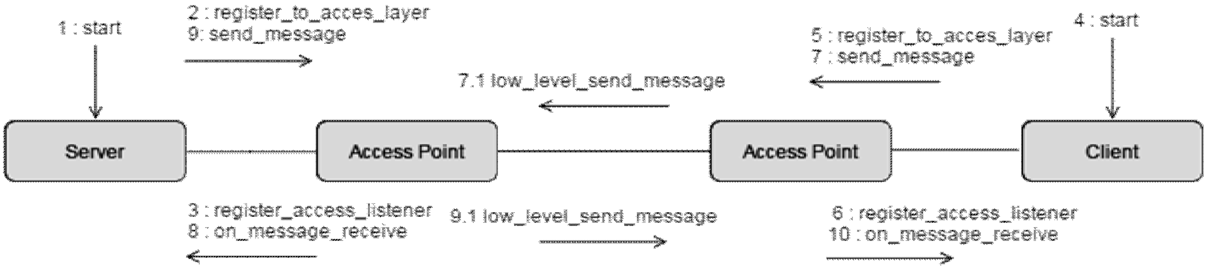

1 : start
2 : register_to_acces_layer
9: send_message
7.1 low_level_send_message
5 : register_to_acces_layer
7 : send_message
4 : start
Server
Access Point
Access Point
Client
3 : register_access_listener
8 : on_message_receive
9.1 low_level_send_message
6 : register_access_listener
10 : on_message_receive

Fig. 10. UML communication diagram of the components communication process

This diagram illustrates both server and client initialization and communication initiation by the server. The messages with the 7.1 and 9.1 numbers does not belong to the protocol and just demonstrate the data transfer via the communication environment, however, they are shown in the diagram for the completeness of the communication process illustration.

D. Verification of the Protocol Realization SM

To verify the software means developed, we shall use the positive test created using the *GTests* [46] – a unit testing library for the *C*++ programming language. The test scenario is as follows:

– creating two communication environments, i.e. the server-client and the client-server ones, on the basis of a sync queue;

– creating the server and the dummy client that will analyze the server output messages;

– creating the access points for both the client and the server and connecting them by the relevant communication environments;

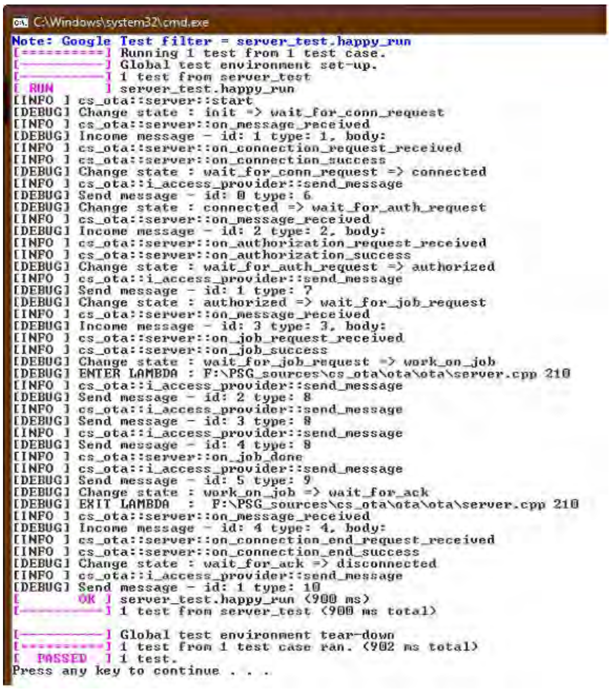
– server registering at the access point;

– creating the separate flow that will generate the client's messages;

– executing the flow;

– comparing the server output messages with expected ones in the dummy client.

The results of the test execution reflected in the OS *Windows* command line are presented in Fig. 11.



```
Note: Google Test filter = server_test.happy_run
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from server_test
[ RUN      ] server_test.happy_run
[INFO ] cs_ota::server::start
[DEBUG] Change state : init => wait_for_conn_request
[INFO ] cs_ota::server::on_message_received
[DEBUG] Income message - id: 1 type: 1, body:
[INFO ] cs_ota::server::on_connection_request_received
[INFO ] cs_ota::server::on_connection_success
[DEBUG] Change state : wait_for_conn_request => connected
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 0 type: 6
[DEBUG] Change state : connected => wait_for_auth_request
[INFO ] cs_ota::server::on_message_received
[DEBUG] Income message - id: 2 type: 2, body:
[INFO ] cs_ota::server::on_authorization_request_received
[INFO ] cs_ota::server::on_authorization_success
[DEBUG] Change state : wait_for_auth_request => authorized
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 1 type: 7
[DEBUG] Change state : authorized => wait_for_job_request
[INFO ] cs_ota::server::on_message_received
[DEBUG] Income message - id: 3 type: 3, body:
[INFO ] cs_ota::server::on_job_request_received
[INFO ] cs_ota::server::on_job_success
[DEBUG] Change state : wait_for_job_request => work_on_job
[DEBUG] ENTER LAMBDA : F:\PSG_sources\cs_ota\ota\ota\server.cpp 210
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 2 type: 8
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 3 type: 8
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 4 type: 8
[INFO ] cs_ota::server::on_job_done
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 5 type: 9
[DEBUG] Change state : work_on_job => wait_for_ack
[DEBUG] EXIT LAMBDA  :  F:\PSG_sources\cs_ota\ota\ota\server.cpp 210
[INFO ] cs_ota::server::on_message_received
[DEBUG] Income message - id: 4 type: 4, body:
[INFO ] cs_ota::server::on_connection_end_request_received
[INFO ] cs_ota::server::on_connection_end_success
[DEBUG] Change state : wait_for_ack => disconnected
[INFO ] cs_ota::i_access_provider::send_message
[DEBUG] Send message - id: 1 type: 10
[       OK ] server_test.happy_run (900 ms)
[----------] 1 test from server_test (900 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (902 ms total)
[  PASSED  ] 1 test.
Press any key to continue . . .
```

Fig. 11. Illustration of the server inner states transitions in the process of communication with the client

Here, using the logging component, we show the transitions of the server inner states according to the input messages arrival from the client. In accordance with the protocol, the *Init* state is the server initial state after initialization. Then, using the *start* method, the server transits into the *wait_for_conn_request* state (this transition is illustrated in the command line with the *[DEBUG] Change state: init => wait_for_conn_request* string appearance). When the server receives a message (this is evidenced by appearing the strings [*INFO] cs_ota::server::on_message_received* and *[DEBUG] Income message – id: 1 type: 1, body:*) in the command line, the server transits into the *Connected* state and so on according to the protocol.

X. CONCLUSIONS

Based on the methodological and technological approaches, namely the method of self-configuring of the computer system with reconfigurable logic, the Software as a Service model and the Internet of Things technology, the method of the computer devices automatic synthesis in the RPH of the CPS smart sensors has been developed. This method, contrary to the available ones, takes into account the specific features of automatic generation of the application-specific processors soft cores, their compilation and logic synthesis, and makes a basis for developing the relevant algorithmic base and corresponding software means. Summarizing, the proposed method allows the computer devices automatic synthesis to be realized as the service for the FPGA-based smart sensors of CPS.

The client-server protocol of information exchange between the RHPs of the CPS measuring and computing nodes has been developed for the computer devices automatic creation in them that, unlike the available information exchange protocols, takes into account the specific features of the automatic generation of the computer device soft cores codes, their compilation and logic synthesis, and does not depend on the communication environment. In accordance with this protocol, the technical requirements for implementation have been formulated and the principles of design and main algorithms of the program interface operation for information exchange between the RHPs of the CPS measuring and computing nodes have been developed to provide the service on the computer devices automatic creation in them. The protocol proposed could be used in future not for the computer device automatic creation only, but for solving other ‘hot’ CPS-related problems as well, because it operates with the abstract notion of ‘task’ that may me modified depending on the purpose.

The data packet format that will operate with the packets for the computer devices automatic creation in the RHPs of the CPS measuring and computing nodes has been developed. The use of the above format will allow the information related to the smart-sensors operation algorithms, FPGAs and their configurations etc., to be transmitted by means of a relevant communication protocol, whereas the fulfillment of the requirements to the program interface that realizes the information exchange protocol will ensure its reliable operation and will give a possibility to work with any necessary resources by providing it with the appropriate flexibility.

The results of the relevant modeling have been presented, and the software means for realizing the protocol of information exchange between the RHPs of the CPS smart sensors for the computer device automatic synthesis in them have been implemented and tested.

The scientific results presented in this paper were obtained within the framework of a research project entitled “*Integration of methods and means of information measuring, automation, processing and protection in the cyber-physical systems basis*” (state registration number 0115U000446, code: DB/KIBER, project term: 01.01.2015–31.12.2017) supported financially by the Ministry of Education and Science of Ukraine.

REFERENCES

[1] Melnyk A. Cyber-physical systems: design issues and development areas / A. Melnyk // Lviv Polytechnic National University Journal “Computer systems and networks”. – 2014. – No. 806. – P. 154–161.

[2] IEEE Std 802.15.4TM 2011, IEEE Standard for Local and metropolitan area networks Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). Revision of IEEE Std 802.15.4-2006, Approved 14 August 2012 by American National Standards Institute.

[3] IEEE P802.11i/D10.0. Medium Access Control (MAC) Security Enhancements, Amendment 6 to IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications. April, 2004.

[4] A Survey on FPGA-Based Sensor Systems: Towards Intelligent and Reconfigurable Low-Power Sensors for Computer Vision, Control and Signal Processing. Gabriel J. García, Carlos A. Jara, Jorge Pomares, Aiman Alabdo, Lucas M. Poggi, Fernando Torres. Sensors (Basel) 2014 Apr; 14(4): 6247–6278. Published online 2014 Mar 31. doi: 10.3390/s140406247.

[5] Moreno-Tapia S. V., Vera-Salas L. A., Osornio-Rios R. A., Dominguez-Gonzalez A, Stiharu I, Romero-Troncoso RJ. A Field Programmable Gate Array-Based Reconfigurable Smart-Sensor Network for Wireless Monitoring of New Generation Computer Numerically Controlled Machines. Sensors. 2010; 10(8):7263-7286.

[6] Vera-Salas L. A., Moreno-Tapia S. V., Garcia-Perez A., Romero-Troncoso R. J., Osornio-Rios R. A., Serroukh I., Cabal-Yepez E. FPGA-Based Smart Sensor for Online Displacement Measurements Using a Heterodyne Interferometer. Sensors. 2011; 11(8):7710–7723.

[7] Wang Y., Bermak A., Boussaid F. FPGA Implementation of Compressive Sampling for Sensor Network Applications. Proceedings of the 2010 2nd Asia Symposium on Quality Electronic Design (ASQED), Penang, Malaysia, 3–4 August 2010; P. 5–8.

[8] Kaddachi M., Soudani A., Nouira I., Lecuire V., Torki K. Efficient Hardware Solution for Low Power and Adaptive Image-Compression in WSN. Proceedings of the 2010 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Athens, Greece, 12–15 December 2010; pp. 583–586.

[9] Chefi A., Soudani A., Sicard G. Hardware Compression Solution Based on HWT for Low Power Image Transmission in WSN. Proceedings of the 2011 International Conference on Microelectronics, Hammamet, Tunisia, 19–22 December 2011; P. 1–5.

[10] Sun Y., Li L., Luo H. Design of FPGA-Based Multimedia Node for WSN. Proceedings of the 2011 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), Wuhan, China, 23–25 September 2011; P. 1–5.

[11] Tanaka S., Fujita N., Yanagisawa Y., Terada T., Tsukamoto M. Reconfigurable Hardware Architecture for Saving Power Consumption on a Sensor Node. Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Sydney, Australia, 15–18 December 2008; P. 405–410.

[12] Khursheed K., Imran M., Malik A., O'Nils M., Lawal N. Exploration of Tasks Partitioning between Hardware Software and Locality for a Wireless Camera Based Vision Sensor Node. Proceedings of the 2011 6th International Symposium on Parallel Computing in Electrical Engineering (PARELEC), Luton, UK, 3–7 April 2011; P. 127–132.

[13] Kwok T. T. O., Kwok Y. K. Computation and Energy Efficient Image Processing in Wireless Sensor Networks Based on Reconfigurable Computing. Proceedings of the 2006 International Conference on Parallel Processing Workshops, Columbus, OH, USA, 14–18 August 2006; P. 8–50.

[14] Pham D. M., Aziz S. FPGA Architecture for Object Extraction in Wireless Multimedia Sensor Network. Proceedings of the 2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Adelaide, Australia, 6–9 December 2011; P. 294–299.

[15] Al-Somani T., Houssain H. Implementation of GF(2m) Elliptic Curve Cryptoprocessor on a Nano FPGA. Proceedings of the 2011 International Conference on Internet Technology and Secured Transactions (ICITST), Abu Dhabi, UAE, 11–14 December 2011; P. 7–12.

[16] Hämäläinen P., Hännikäinen M., Hämäläinen T. D. Review of Hardware Architectures for Advanced Encryption Standard Implementations Considering Wireless Sensor Networks. Proceedings of the 7th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, 16–19 July 2007; P. 443–453.

[17] Antonio de la Piedra, An Braeken, Abdellah Touhafi. Sensor Systems Based on FPGAs and Their Applications: A Survey. Sensors 2012, 12(9), 12235–12264.

[18] Melnyk A., Melnyk V. “Personal Supercomputers: Architecture, Design, Application”. Lviv Politechnic National University Publishing. – 2013. – 516 pp.

[19] Melnyk A., Melnyk V. “IP Cores Design Methodology”. Lviv Polytechnic National University Journal “Computer systems and networks”. – 2002. – No. 463. – P. 3–9.

[20] Melnyk A., Melnyk V., “Self-Configurable FPGA-Based Computer Systems”, Advances in Electrical and Computer Engineering, vol. 13, no. 2, pp. 33–38, 2013, doi:10.4316/AECE.2013.02005. [Online]. Available: http://www.aece.ro/abstractplus.php?year=2013&number=2&article=5

[21] Melnyk V. Self-Configurable FPGA-Based Computer Systems: Basics and Proof of Concept. Scientific-Technical Journal “Advances in Cyber-Physical Systems”. Vol. 1, No. 1, 2016. – pp. 37–47.

[22] Paul Gil. “What Is 'SaaS' (Software as a Service)?”. About. [Online]. Available: http://netforbeginners.about.com/od/s/f/what_is_SaaS_software_as_a_service.htm. Retrieved 19 December 2016.

[23] “Definition of: SaaS”. PC Magazine Encyclopedia. Ziff Davis. [Online]. Available: http://www.pcmag.com/encyclopedia/term/56112/saas. Retrieved 19 December 2016.

[24] “Software as a Service (SaaS)”. Cloud Taxonomy. Open crowd. [Online]. Available: http://cloudtaxonomy.opencrowd.com/taxonomy/software-as-a-service/. Retrieved 19 December 2016.

[25] “Internet of Things Global Standards Initiative”. ITU. [Online]. Available: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx. Retrieved 26 June2015.

[26] Brown, Eric (13 September 2016). “Who Needs the Internet of Things?”. Linux.com. [Online]. Available: https://www.linux.com/news/who-needs-internet-things. Retrieved 23 October 2016.

[27] Melnyk V., Stepanov V., Sarajrech Z., “System of load balancing between host computer and reconfigurable accelerator”, Proceedings “Computer systems and components” of Tchernivtsi National University. – Tchernivtsi. 2012. T. 3. Ed. 1. P. 6–16.

[28] Chameleon – the System-Level Design Solution. [Online]. Available: http://intron-innovations.com/?p=sld_chame.

[29] Melnyk A. Chameleon – Application-Specific Processors High-Level Synthesis Environment / A. Melnyk, A. Salo, V. Klymenko, L. Tsyhylyk, A. Yurchuk // Scientific and Technical Journal of National Aerospace University “Kharkiv Aviation Institute”, Kharkiv, 2009. – No. 5. – P. 189–195.

[30] Agility Compiler for SystemC. Electronic System Level Behavioral Design & Synthesis Datasheet. 2005. [Online]. Available: http://www.europractice.rl.ac.uk/vendors/agility_compiler.pdf.

[31] Handel-C Language Reference Manual For DK Version 4. Celoxica Limited, 2005. – 348 p.

[32] C-to-FPGA Tools form Impulse Accelerated Technologies. Impulse CoDeveloper C-to-FPGA Tools. [Online]. Available: http://www.impulseaccelerated.com/products_universal.htm.

[33] Clive Maxfield, EE Times. “WebPACK edition of Xilinx Vivado Design Suite now available”. Dec 20, 2012.

[34] Clive Maxfield, “Latest and greatest Quartus II design software from Altera”, EETimes, November 7, 2011.

[35] Transmission Control Protocol. Darpa Internet Program. Protocol Specification. September, 1981.

[36] Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. International Standart. ISO/IEC 7498-1. Second edition. 1994-11-15.

[37] The Requirements Engineering Handbook. Ralph R. Young, Artech House, Boston, London, 2004.

[38] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.

[39] “The Single Responsibility Principle”, Robert C. Martin (“Uncle BOB”), http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod.

[40] “Package principles”, Robert C. Martin (“Uncle BOB”), http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod.

[41] About CMake, https://cmake.org/overview/.

[42] About Winsock, https://msdn.microsoft.com/en-us/library/windows/desktop/ms737523(v=vs.85).aspx

[43] UNIX Network Programming Volume 1, Third Edition: The Sockets Networking API, W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, Addison Wesley, 2003.

[44] “The Dependency Injection Principle”, Robert C. Martin (“Uncle BOB”), http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod.

[45] “Design Patterns: Elements of Reusable Object-Oriented Software”, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison-Wesley, 1994, p 395, ISBN 0-201-63361-2.

[46] Google Test, https://github.com/google/googletest.

Viktor Melnyk is a professor of the Department of Information Technologies Security in Lviv Polytechnic National University, Ukraine. He was awarded with the academic degrees of philosophy doctor in 2004, and doctor of technical sciences in 2013 in Lviv Polytechnic National University. He has scientific, academic and hands-on experience in the field of computer systems research and design, proven contribution into IP Cores design methodology and high-performance reconfigurable computer systems design methodology. He is experienced in computer data protection, including cryptographic algorithms, cryptographic processors design and implementation, wireless sensor network security. Mr. Melnyk is an author of more than 80 scientific papers, patents and monographs.

Ivan Lopit received B.Sc. degree in information security from Lviv Polytechnic National University, Ukraine, in 2012, and M.Sc. degree in 2013. He is currently pursuing Ph.D. degree in computer systems and components in Lviv Polytechnic National University. Besides he is a Senior Software Developer at Intellias Company based in Lviv. He is the author of 4 articles in scientific journals and 7 publications in conferences proceedings. His research interests include high-performance computing, cyber-physical systems, memory compression, hardware and software optimization techniques.

Andrii Kit received B.Sc. degree in information security from Lviv Polytechnic National University, Ukraine, in 2012, and M.Sc. degree in 2013. He is currently pursuing Ph.D. degree in computer systems and components in Lviv Polytechnic National University.

From 2015 to 2016, he was an Assistant Professor with the Department of Information Technologies Security in Lviv Polytechnic National University. He is the author of 10 publications in scientific journals and conferences proceedings. His research interests include cyber-physical systems, software development for embedded systems, computer systems and networks, ultra-scale computer systems, fundamental study of operating systems and networking.