Available online at http://scik.org

J. Math. Comput. Sci. 11 (2021), No. 1, 520-542

https://doi.org/10.28919/jmcs/5152

ISSN: 1927-5307

DEVELOPMENT OF A CYBER DESIGN MODELING DECLARATIVE LANGUAGE FOR CYBER PHYSICAL PRODUCTION SYSTEMS

IGOR NEVLIUDOV[1], VLADYSLAV YEVSIEIEV[1], JALAL HASAN BAKER[2], M. AYAZ AHMAD[2,*], VYACHESLAV LYASHENKO[3]

[1]Department of Computer-Integrated Technologies, Automation and Mechatronics, Kharkiv National University of Radio Electronics, Ukraine

[2]Department of Physics, Faculty of Science, University of Tabuk, Saudi Arabia

[3]Department Informatics, Kharkiv National University of Radio Electronics, Ukraine

Abstract. This paper considers the issues of software development automation for cyber-physical production systems within the framework of the Smart Manufacturing (SM) and Industry 4.0 concepts. The development of cyber-physical production systems (CPPS) for each enterprise is an individual task that takes into account the specifics of the production process, the requirements for data visualization at each level from Supervisory Control And Data Acquisition (SCADA) to Enterprise Resource Planning System (ERP) in a single information space using different technologies for processing and storing data. To ensure all the requirements that are specified by the customer in the technical requirement (TR), the authors propose to automate the process of developing a cyber-physical production systems Human-Machine Interface (HMI) prototype at an early stage of drafting the terms of reference together with the customer, which will make it possible to take into account all the requirements for visualizations information. Based on the experience gained in the development of cyber-physical production systems, the authors have developed a declarative cyber-design modeling language based on parameters mathematical description of the and Graphical User Interface (GUI) elements events, which allows to simplify the

[*]Corresponding author

E-mail address: mayaz.alig@gmail.com

Received October 31, 2020

process of HMI cyber-physical production systems development at an early stage by generating fragments of program code for object-oriented programming languages.

Keywords: Industry 4.0.; smart manufacturing; cyber-physical production systems; human-machine interface; graphical user interface; modeling language.

2010 AMS Subject Classification: 93A30.

1. INTRODUCTION

Modern high-tech production is not possible without the implementation of Industry 4.0 concepts, the basis of which is the creation of Digital Twins, which are implemented in the form of cyber-physical production systems (CPPS). CPPS is the synthesis of physical (sensors, executing mechanisms) and cybernetic (control programs, monitoring, decision-making, visualizations) components in a single information space [1-5]. Considering the cybernetic component as a set of software and modules that, using the Human-Machine Interface (HMI), to interact with users at levels from Supervisory Control And Data Acquisition (SCADA) to Enterprise Resource Planning System (ERP) [6-11]. A critical analysis of publications showed that the development of CPPS software is based on the following methodologies: Rapid Application Development (RAD), Rational Unified Process (RUP), Disciplined Agile Delivery (DAD), is partially possible, but none of the listed methods does not allow automation of CPSS cyber part development process [12-15]. Based on their own experience in creating CPPS, the authors developed a declarative HMI cyber-design modeling language based on the properties of object-oriented programming languages GUI elements.

2. DEVELOPMENT OF A GUI-BASED HMI MODELING LANGUAGE FOR CYBER-PHYSICAL PRODUCTION SYSTEMS

2.1 Definition of the CPPS cyber design modeling language keywords

Let us define the concept of a cyber design modeling language for CPPS as a declarative (non-procedural) language, the purpose of which is to define and describe terminologies based on models and relationships between metadata and domain data and ways of transforming them. Within the framework of the research data, we define the following keywords:

$Form$ *(Windows Form)* – some dedicated and uniquely identified part of the subject area. Within the framework of these studies, it has the following properties:

$$Form = Form_1^{master} \cup Form_n^{slave} \qquad (1)$$

$$Form_1^{master} = \bigcup_{n=2} Form_n^{slavr}$$

$ParameterForm$ is a types and methods set of describing the subject area properties, highlighted and grouped according to some characteristics, as well as identified by name. Purpose is a description of the parameters necessary and sufficient for displaying and modeling a visual presentation $Form$.

$$\begin{aligned}&((parameter_1,\ldots,parametr_p) \in ParameterForm_z^1) \in Form_1^{master} \equiv \\ &\equiv ((parameter_1,\ldots,parameter_p) \in ParameterForm_z^1) \in Form_2^{slave}\end{aligned} \qquad (2)$$

$EventForm$ is an event or group of events (action) that can occur (have already occurred or will occur) with the subject area at some moment or time interval. It can be identified by time (necessity) and the object to which the event belongs. Only one user-initiated event can occur with one object at a time.

$$\begin{aligned}&((event_1,\ldots event_e) \in EventForm_c^1) \in Form_1^{master} \equiv \\ &\equiv ((event_1,\ldots event_e) \in EventForm_c^2) \in Form_2^{slave}\end{aligned} \qquad (3)$$

$ParameterElement$ are the types and ways of describing GUI elements properties, single or grouped according to some characteristics and identified by name. Purpose is a description of the parameters necessary and sufficient for the presentation and modeling of the element visual representation within a single information object.

$$((parameter_1,\ldots,parametr_p) \in ParameterElement_h^1) \in ElementForm_t^1 \qquad (4)$$

$ValueElement$ is the value assigned to the type and way of describing the GUI element properties. Purpose is the assignment of a specific value (integer, linguistic, boolean) to a type or method of describing parameters, depending on the functional features and the implementation of a visual intuitive interface for each part of the subject area.

$$\exists(ValueElements_1, ValueElements_2, \ldots ValueElements_y) \in parameter_p \tag{5}$$

$EventElement$ is an event, a group of events or a condition that can occur (have already occurred or will occur) with a GUI element that performs a specific function at some point or time interval.

$$\exists! LingusticVariable_w \in LingusticVariable \forall event_e \in EventElement_r \tag{6}$$

$LingusticVariable$ is one variable named (in the natural language of the system) logical description of actions when events occur. Such descriptions can be grouped according to a number of characteristics. Purpose is the assignment to a class of an event or a single event of a linguistic intuitively understandable user of the variable developed model to describe reactions when an event occurs.

$$\begin{aligned} &\exists! ContainerSolutions_d \in ContainerSloution = \\ &LingusticVariable_w \in LingusticVariable \end{aligned} \tag{7}$$

$ContainerSolutions$ is a named description of reactions when an event or a group of events occurs at a certain point in time to an element (group of elements) or subject area. Has a rigidly structured structure (depending on the high-level programming language and development environment), which is necessary to achieve the development goal or is indicated in the TR.

$$\begin{aligned} &cod_{lw} \in ContainerSloution_d = \\ &LingusticVariable_w \in LingusticVariable \end{aligned} \tag{8}$$

2.2 Development of specifications for the CPPS cyber-design modeling language

The following data model language specification is proposed:

- allowed **alphanumeric** characters that are supported by development environments for high-level programming languages and correspond to the ASCII code table: + - \ . , ! “ < > = ( ) $ % & ~ * _ & @ space ; { };

- **key words**: basic concepts in the form of words reserved in the developed NM and are used to describe key features are described above.

- **identifiers,** used to indicate the following features:

- attribute of parameters, events belonging to a domain or non-domain type: $domen$, not_domen. Domains of the corresponding characteristics (values) belonging to the enumerated (list) type, which can be selected from a pre-formed list. An example is how some parameters $ParameterForm$, display $Form$, which can take values $true$ or $false$, parameter $Align$ which is inherent $dom(parameter^{z}_{p\_список})$ and $dom(parameter^{h}_{p\_список})$ from expression:

$$ParmeterForm^{1}_{z} = \{parameter^{z}_{1}, parameter^{z}_{2}, ..., parameter^{z}_{p}\};$$
$$ParmeterElement^{1}_{h} = \{parameter^{h}_{1}, parameter^{h}_{2}, ..., parameter^{h}_{p}\};$$
$$EventForm^{1}_{c} = \{event^{c}_{1}, event^{c}_{2}, ..., event^{c}_{e}\};$$
$$EventElement^{1}_{r} = \{event^{r}_{1}, even^{r}_{2}, ..., event^{r}_{e}\};$$
$$ParameterForm_{z} = dom(parameter^{z}_{p\_спис}); \tag{9}$$
$$ParameterElement_{h} = dom(parameter^{h}_{p\_спис});;$$
$$EventForm^{1}_{c} = dom(event^{c}_{e\_спис});$$
$$EventElement^{1}_{r} = dom(event^{r}_{e\_спис});$$

where $dom(parameter^{z}_{p\_спис})$, $dom(parameter^{h}_{p\_спис})$, $dom(event^{c}_{e\_спис})$, $dom(event^{r}_{e\_спис})$ are the domains of the corresponding characteristics (values) belonging to the enumerated (list) type, which are selected from a pre-formed list, which can take on the values fixed by the CPPS development environment specified in the TR.

- data type of values ($value$), which defines the characteristics of the parameters $ParameterForm$ and $ParametrElement$ (text, boolean, integer, integer negative, text word combination).

- basic concepts that make it possible to link events $ElementForm$ and $EventElement$ containing a set of defined $event$, belonging to a specific visual graphic with $ContainerSolution\,(cod)$ through $LingusticVariable(name)$.

As you can see, unlike keywords, the proposed identifiers can theoretically be redefined, but this gives the errors possibility, as a result of which the identifiers listed above are included in the fixed key layers dictionary.

- **literals,** a specific set of values that are not represented by an identifier.

String literals are represented as a sequence of allowed characters with different types of writing (uppercase and lowercase) letters. E.g., *name_form*, the name of the form that is used in the parameters $Caption$, $Name$ etc., And also assigning a unique name. $(name)$ for each $LingusticVariable$, which contains a certain piece of program code. An example of "save in the database", "calculate the result", etc., which are set by the end user for the convenience of using the developed language.

Algebraic literals are a description of simple logical operations like $True, False$, which allow you to set values $(value)$ this or that parameter $(parameter)$, owned by $ParametrElement$, $ParameterForm$, necessary and sufficient to describe the properties of developed CPPS visual elements or software modules in accordance with the requirements of the TR.

Reserved literals represent a word, phrase or abbreviation, which makes it possible to select one or another property of the parameter necessary to achieve the requirements specified in the technical specification by the customer. An example would be a shape property $WindowsState$ in the RadStudioXE6 development environment, which can select the following reserved word abbreviations as $wsNormal, wsMinimized, wsMaximized$, that is, at the initial launch, the developer can set the display of the form. $wsNormal$ – the display by default, in the form in which it was created at the design stage, $wsMinimized$ – the form is displayed minimized on the taskbar, $wsMaximized$ – when launched, the form expands to fit the entire desktop.

Reserved literals can be shared between $ParameterForm$ and $ParametrElement$, and also specialized, that is, belong to a certain visual form that describes the specifics of a particular element. But, it should be noted that reserved characters for determining the values of one or another visual components parameter that have the same purpose, but can perform different specified functions and process events in the same development environment.

Types of presented values, which contain some parameters $ParameterForm$ и $ParametrElement$, are permissible in the field of application:

Integer data type (integer) allows you to assign a parameter $ParameterForm$ or/and $ParametrElement$ a defined and required digital value of the dimension or coordinates of the visual element placement relative to $Form$. Used primarily to describe visual graphical elements. The smallest logical element of a two-dimensional digital image in a bitmap (pixel). The length of the line depends on the screen resolution and the technical specification requirements are presented by the customer to the developer.

Negative integer allows you to assign a parameter to a specific value within the ($-1,1,2,3,...,n$) range, which belongs exclusively to $ParametrElement$ and describes numbering in this context:

-1 – no numbering, parameter is not used;

– $1,2,3,...,n$ – numbering of a graphic image (icon) that belongs to a certain parameter ($parameter$) for $ElementForm$.

Text / linguistic (char) allows you to assign to a parameter logically ordered values of symbols that contain explanations necessary for the user or the graphic elements name necessary for the convenience of working with CPPS. Also, this type of value representation is used to set a specific name $LingusticVariable$, which is assigned to the event $EvenForm$, $EventElement$.

Logical (boolean) – can take only two values – true (true) or false (false) and play the role of a switch for using one or another parameter in $ParameterForm$ and $ParametrElement$.

Text phrase (enumerated type) is the the data type specified by the list in the domain form presented above allows you to specify a list of reserved words in the development environment or abbreviations that can be accepted by one or another $parameter$ for $ParameterForm$ and $ParametrElement$.

Separators are the symbols highlighted the basic elements of the developed modeling language syntactic structure.

$<Form>$ (angle brackets $Form$) – is used to indicate a keyword that indicates the beginning of $Form$ meta description in modeling language design.

$</Form>$ (slash angle brackets $Form$) – is used to indicate a keyword that indicates the completion of the $Form$ meta description in modeling language design.

For the proposed keyword construction, at the beginning and end of $Form$ meta description the following restrictions are imposed: $Form$ name can be numbered as $Form1$ or literal definition for example $Form\_master$ or $Form\_add\_operat$. In this case, the key word for the beginning of $Form$ meta description must coincide with the key word for the end of the meta description in modeling language design. If this design requirement is not met, the interpreter of the modeling language will not be able to perceive the content as a meta-description of all the necessary parameters and events inherent in this $Form$.

{ (opening curly brace) is the required beginning character of the $Form$ and $ElementForm$ meta description line.

} (closing curly brace) is required meta description line terminator of $Form$ and $ElementForm$.

# (number sign) – after this symbol, the structure of the modeling language design interpreter perceives the beginning of the user interface graphical visual elements description ($ElementForm$).

/# (slash number sign) – after this combination of symbols, the YM interpreter considers that the description of the user interface graphical visual elements ($ElementForm$) is finished.

/ (slash) – used to define hierarchies of visual graphical elements meta description ($ElementForm$) according to the CPPS construction tree, and is applied internally $\#$ $/\#$ meta descriptions of $Form$. $ElementForm1/ElementForm2$ must be understood as $ElementForm2$ be inside $ElementForm1$ and is an integral part of it.

[ ] square brackets are used for the task of meta description of required parameters and events $ParameterForm$, $EvenForm$, $ParametrElement$, $EventElement$.

; (semicolon) - a mandatory symbol of modeling language structures, which shows that for a given $parameter$ or $event$ assignment of $value$ and $name$ respectively, completed, applied internally [ ].

, , (comma-separated list) – used to list names $parameter$ for $ParameterForm$, $ParametrElement$, as well as $event$ for $EvenForm$, $EventElement$ provided that for a set of several $parameter$ or $event$ values $value$ and $name$ respectively, the same and applied internally [ ].

= (equal sign) – assigns to $parameter$ a specific data type value and is used to indicate an event ($event$) of certain $name$ from $LingusticVariable$ which contains a link to cod or a fragment of it in $ContainerSolution$. It should be noted that, depending on the context (logic and meaning of the actions performed), this sign can also be interpreted as an assignment instruction, according to which the value belonging to it is determined for the specified basic parameter.

Comments – all characters and lines written inside this construction are ignored by the modeling language (ML) interpreter and are perceived as comments. Alphanumeric characters of national alphabets supported by the operating system and development environment are allowed. The limitation for comments is that the sequence should not exceed 255 (FF) characters.

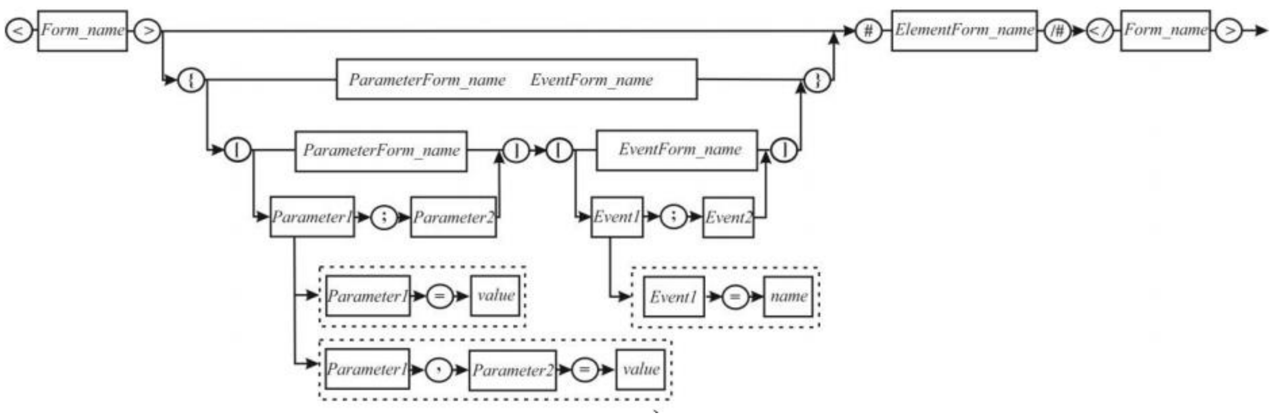
?** (question mark with two asterisks) shows that the given characters are followed by a comment, which is ignored by the ML interpreter.

**? (two asterisks and a question mark) – shows that after the given characters, the comment ends and then the text that is not ignored by the ML interpreter.

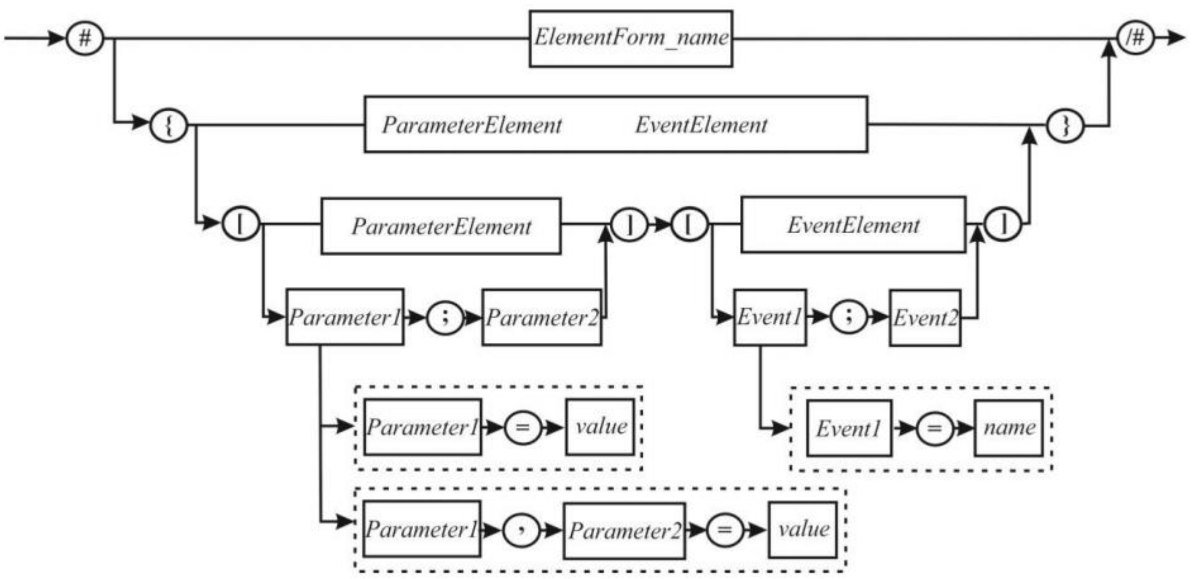
To adapt the developed syntax for ML describing, we proposed to use the Beckus-Naur form. The rationale for this choice was that the extended Backus-Naur form is used to describe context-free grammars and makes it possible to simplify and shorten the description [16-18]. The extended Backus-Naur form is described in the international standard ISO / IEC -14977 [19]. Analysis of ISO / IEC-14977 showed that the extended Backus-Naur form makes it possible to develop an intuitively simple and adaptive formal language for representing and describing the necessary data for CPPS development based to object-oriented programming approaches.

2.3 CPPS cyber-design modeling language syntax diagram development

Based on the proposed above specification of the language matamodel, the authors propose the following syntax diagram, which is shown in Figure 1.



a)



b)

Figure 1. Syntactic diagram of the developed ML (a, b)

The syntax diagram of the values representation types that can belong to identifiers proposed in this study is shown in Figure 2.

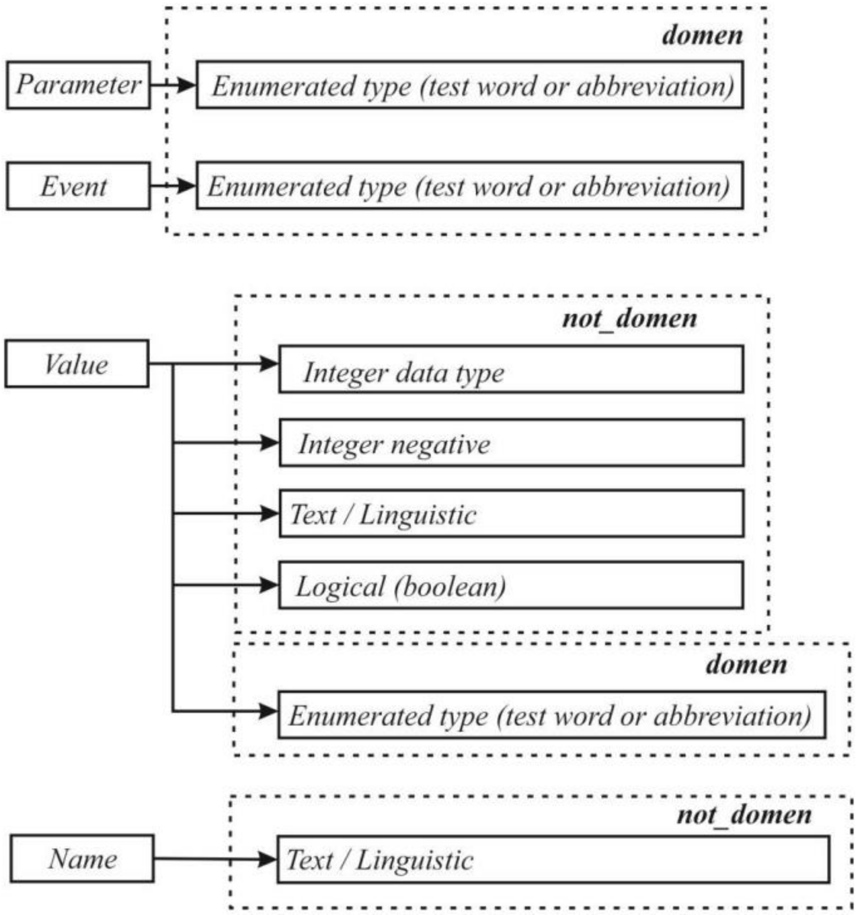


Figure 2. Syntax diagram of identifier values representation types

As you can see from Figure 2, the identifiers $Parameter$, $Event$ refer to $domen$ (list) type and is represented as a text word or abbreviation that refers to $ParameterForm\_name$ and $EvenForm\_name$, as well as $ParameterElement$ and $EvenElement$ according to Figure 1. Parameter ($parameter$) and events ($event$) list which belong $ParameterForm\_name$, $EvenForm\_name$ in the framework of one development environment are permanent and not changeable. For parameter ($parameter$) and events ($event$) list which belong to $ParametrElement$, $EventElement$ accordingly, it is an equal limitation that these visual graphical elements have the same purpose within the development environment. It is worth noting that this type includes $value$ for $ParametrElement$ and $ParameterForm\_name$, which contains a text word or abbreviation reserved by the IDE.

Identifiers $value$ and $name$ belongs to not_domen (non-list) type. This is justified by the fact that the values $value$ can be set by the developer depending on the requirements put forward by the developed CPPS. For identifier $name$, which is included in $LingusticVariable$ the name that refers to $ContainerSolution$ containing the necessary fragment or part of the program code (cod), set by the user of the system being developed, taking into account his logical preferences and ease of use.

For ease of reading and presentation of the developed declarative language (Figures 1 and 2), it is necessary that it has the qualities of understanding and reading. This can be achieved using at least three principles of language representation [20], namely was:

- maximally linear;
- short;
- self-documented.

Based on the proposed assumptions and recommendations for the declarative language being developed, the CPPS developer proposes the following type of model language notation style, which makes it possible to simplify and standardize the code.

Example 1

$< Form\_master >$

$\{$?** opening a block for describing parameters and values, as well as events and names $LingusticVariable$ for $Form\_master$ **?

[$parameter1 = value; parameter2, parameter3 = value$]

[$event1 = name; event2 = name$]

$\}$?** closing the block for describing parameters and values, as well as events and names $LingusticVariable$ for $Form\_master$ **?

# “ element name in development environment”

?** opening a block for describing visual graphic elements $Form\_master$ **?

$\{$?** description block $Element1\_Form\_master$ **?

$[\ parameter1 = value; parameter2, parameter3 = value\ ]$

$[\ event1 = name; event2 = name\ ]$

$\}$?** description block closing $Element1\_Form\_master$ **?

$\{$?** description block $Element2\_Form\_master$ **?

$[\ parameter1 = value; parameter2, parameter3 = value\ ]$

$[\ event1 = name; event2 = name\ ]$

$\}$?** description block closing $Element2\_Form\_master$ **?

/# ?** the block of visual graphic elements description closing $Form\_master$ **?

$</Form\_master>$

If it is necessary to implement a hierarchy (construction tree), the membership of visual graphic elements of $ElementForm1/ElementForm2$. The following snippet of the meta description structure is suggested:

Example 2.

# "element name in development environment" ?** opening a block for describing visual graphic elements $Form\_master$ **?

$\{$?** description block $Element1Form\_master$ **?

$[\ parameter1 = value; parameter2, parameter3 = value\ ]$

$[\ event1 = name; event2 = name\ ]$

$\}$?** description block closing $Element1Form\_master$ **?

/ "element name in development environment"

$\{$?** description block $Element2Form\_master$ **?

$[\ parameter1 = value; parameter2, parameter3 = value\ ]$

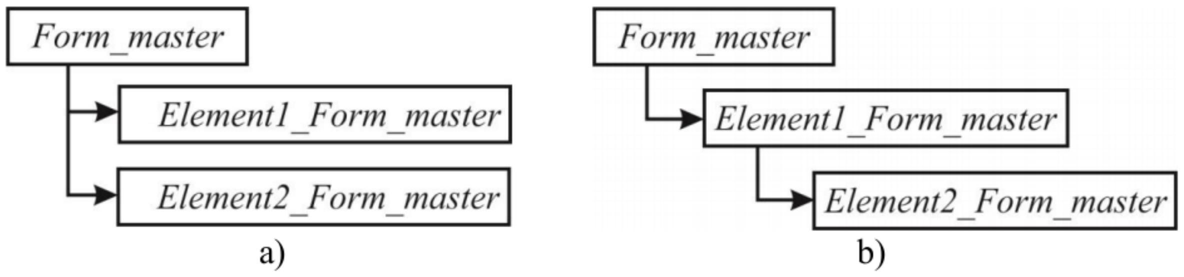
$[\ event1 = name; event2 = name\ ]$

$\}$?** description block closing $Element2Form\_master$ **?

$/\#$?** the block of description of visual graphic elements closing $Form\_master$ **?

Use of “/” (slash) will allow the ML interpreter to determine the degree of a visual element nesting (belonging) to another, that is, to implement a CPPS structure tree in the development environment. Figure 3 shows a graphical structure construction of the construction tree $Form\_master$ of CPPS for example 1 (a) and example 2 (b).

To indicate the corresponding values $(value)$ and names $(name)$, in the examples above, within $LingusticVariable$ for $parameter$ and $event$ respectively after the equal sign $(=)$ the type of value is set, if there are no values or default values reserved by the development environment are used, this parameter is not declared in the meta description (not indicated).



a) $Element1Form\_master$ and $Element2Form\_master$ is equivalent to $Form\_master$;

b) $Element2Form\_master$ belongs to $Element1Form\_master$;

Figure 3 –Graphical representation of the construction tree CPPS structures.

2.4 Experimental research

Example 3 shows a line of meta description of creating an empty form in the RadStudio XE6 environment for VLC Form Application.

Example 3

$<Form1>$

$$\{$$

$$[Caption = example\,1, ClientHeight = 464, ClientWidth = 687, \\ Height = 503, Name = Form\_master, Width = 703] \qquad (10)$$

$$\}$$

$$< / Form1 >$$

Based on the meta description given in 10, a graphical representation of the simplest custom form was generated.

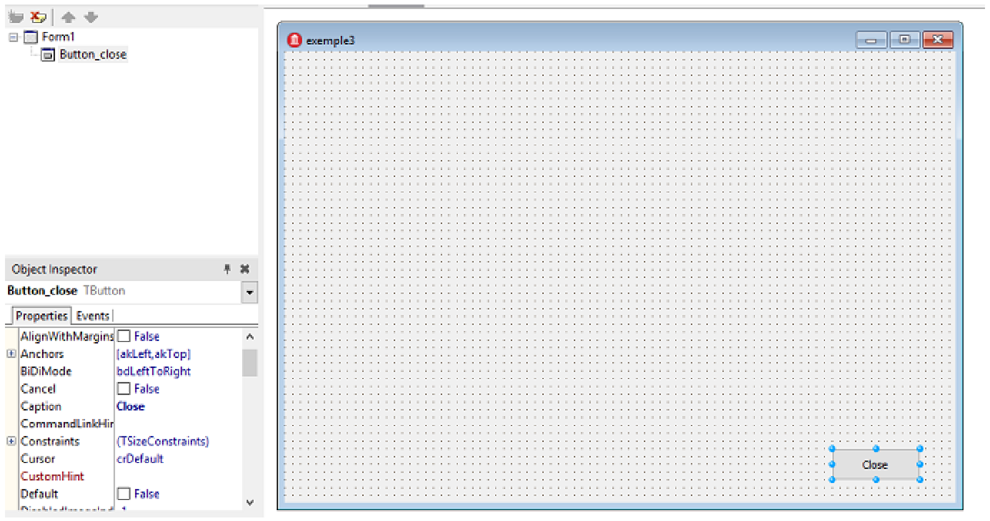
A fragment of the meta description of additional visual graphic elements of the *Standard-Button* type (a custom button that executes a specific event) is presented at 11.

"Buttion_close"

$$[Caption = Close, Height = 33, Top = 408, Left = 560, \\ Name = Buttion\_close, Width = 91] \qquad (11)$$

/#

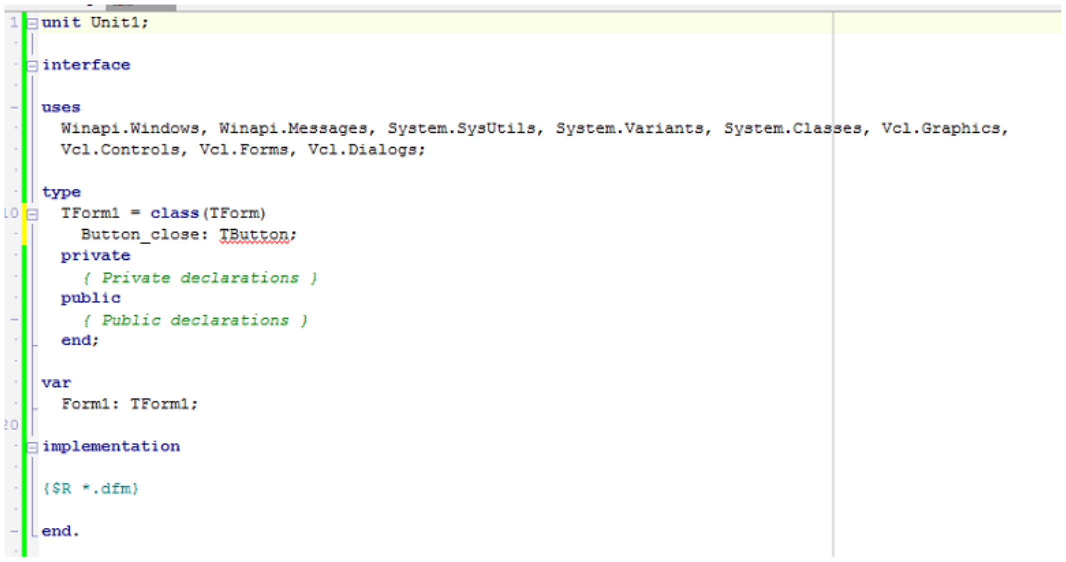
Figure 4 shows an example of implementing a form with a Button element, the meta description of which is given in 10 and 11, respectively.



AlignWithMargins	☐ False
Anchors	[akLeft,akTop]
BiDiMode	bdLeftToRight
Cancel	☐ False
Caption	Close
CommandLinkHir	
Constraints	(TSizeConstraints)
Cursor	crDefault
CustomHint	
Default	☐ False

Figure 4 - Fragment of the development environment with the form and the Button graphic element implementation, obtained on the basis of meta descriptions 10 and 11.

In addition to the implementation of the graphical visual interface shown in Figure 4, based on meta descriptions 10-11, the program code in Pascal was generated, shown in Figure 5.



```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs;

type
  TForm1 = class(TForm)
    Button_close: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Figure 5 - Program code in Pascal, generated on the basis of meta description 10-11.

Each element of the ML description given in the syntax model (10-11) is written in accordance with the syntax diagram shown in Figure 1 and the diagram of the indicators values representation types, is shown in Figure 2. The ML semantic model is a system of values ascribed to constructions and the developed syntactic model of the language (interpretation of constructions). This model is presented in the process of interpreting (parsing) the proposed rules for describing and presenting ML specifications, symbols and their combinations..

Consider the meta description of example 4, (if necessary, implement the attachment (belonging) of one visual element to another as shown in Figure 3-b). In accordance with the proposed syntactic model (Figure 1), the meta description will take the following form:

Example 4

$$\# \ \text{“}GrupBox1\text{”}$$

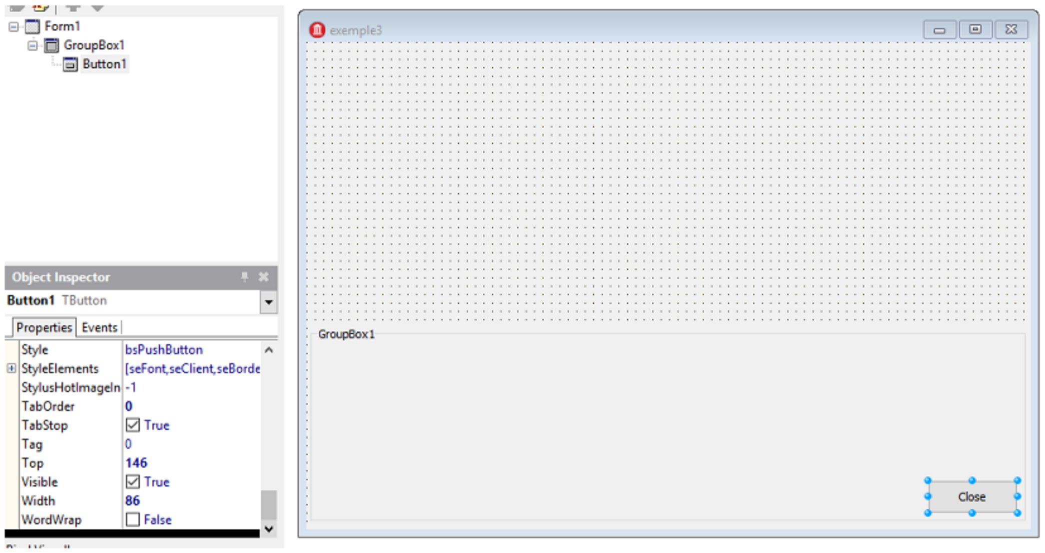
$$[Caption = GroupBox1, Height = 186, Top = 272,$$
$$Left = 4, Name = GroupBox1, Width = 678]$$

$$/ \ \text{“}Buttion1\text{”} \qquad (12)$$

$$[Caption = Close, Height = 32, Top = 146, Left = 585,$$
$$Name = Buttion1, Width = 86]$$

/#

In the development environment, this meta-description allows you to implement the degree of nesting of visual graphic elements in each other and build a "tree" of $Form1$, on the basis of which the user interface is developed in accordance with the technical specification for the CPPS and the algorithm of functioning. Figure 6 shows a fragment of the RadStudio XE6 development environment with a generated user interface in accordance with meta description 12.



Object Inspector
Button1 TButton
Properties | Events

Style	bsPushButton
StyleElements	[seFont,seClient,seBorde
StylusHotImageIn	-1
TabOrder	0
TabStop	☑ True
Tag	0
Top	146
Visible	☑ True
Width	86
WordWrap	☐ False

Figure 6 - Implementation of the $Form1$ interface based on a fragment of the meta description of graphic visual elements 12.

Based on 10-12 meta descriptions and Figure 3, you can set any nesting depth for graphical user interface elements. This makes it possible to implement, using the proposed ML syntax diagram (Figure 1), a CPPS structure of complexity degree, and to simplify the process of developing a visual component based on an object-oriented approach to programming.

Based on the proposed syntax diagram presented in Figure 1, the following event method description is proposed ($event$) for $Form$ and $ElementForm$. Based on example 1, we will give an example of a simple method triggering description on an element $Button$ of event $Onclick$ to execute $LingusticVariable$ with name $Close\_All$.

Example 5

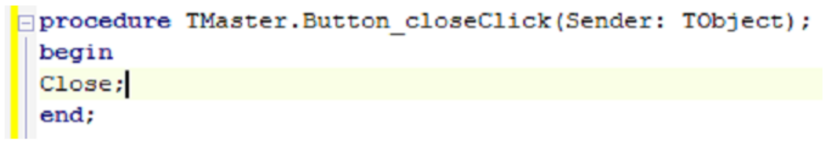
$$
\begin{aligned}
&< Form1 >\\
&\qquad \{\\
&\quad [Caption = example\,1, ClientHeight = 464, ClientWidth = 687,\\
&\quad Height = 503, Name = Form\_master, Width = 703]\\
&\qquad [Caption = Close, Height = 33, Top = 408, Left = 560,\\
&\qquad Name = Buttion\_close, Width = 91, OnClick = Close\_All]\\
&\qquad \}\\
&< / Form1 >
\end{aligned}
\tag{13}
$$

As you can see from the method description in example 5, the developer describes the existence of an event on an element $Buttion$ under the name $Name = Buttion\_close$, $LingusticVariable$ under the name $Close\_All$ at the event $Onclick$. This view allows the developer to implement the following sequence:

$$event \to LingusticVariable \to ContainerSolution \to cod \tag{14}$$

where cod is a a piece of program code in the selected object-oriented language.

As a result of the method description 13 implementation, the developer receives the generated program code shown in Figure 7.



```
procedure TMaster.Button_closeClick(Sender: TObject);
begin
Close;
end;
```

Figure 7 - The result of example program code generation.

Let us consider a fragment of method description example using an example 6 implementations of more complex code constructions that was generated in the design process of the “Automated standardization system “NORMA” (copyright certificate of Ukraine No. 57667 dated 17.12.2014). On the form $Form1$ there is an element $DBGrid\_operac$ to display information from the database. $Properties\_DBGrid\_operace$ the specified binding to the visual dropdown interface $PopupMenu\_operac$ element. It is necessary to generate a program code for deleting the selected record from the database in $DBGrid\_operac$.

Example 6

$$
\begin{array}{l}
< Form1 > \\
\qquad \vdots \\
\qquad \#\ \text{“}DBGrid\_operace\text{”} \\
\qquad \{ \\
\qquad\ \ [DataSurce = Form1.IBDataSet\_Nak\_Operac, \\
\qquad\ \ Height = 120, Left = 344, Top = 24, Width = 320, \\
\qquad\ \ Name = DBGrid\_operace, \\
\qquad\ \ PopupMenu = PopupMenu\_operac] \\
\qquad \} \\
\qquad\quad \vdots \\
\qquad \#\ \text{“}DBGrid\_operace\text{”} \\
\qquad \{ \\
\qquad [Name = PopupMenu\_operac, Items = 28,] \\
\qquad [OnClick = N28Click] \\
\qquad \} \\
\qquad \vdots \\
\qquad \#\ \text{“}N28\text{”}
\end{array}
\qquad (15)
$$

$$\{$$

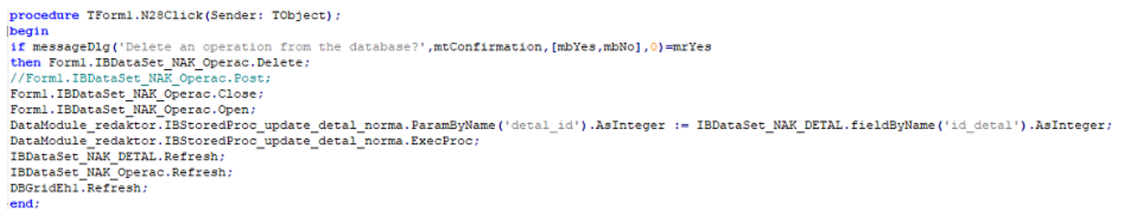
$$[Caption = Delete, Name = 28]$$
$$[OnClick = Delete\_select\_operace]$$

$$\}$$

$$\vdots$$

$$</Form1>$$

An example of the generated program code for implementing an event per element $PopupMenu\_operac$, on single click in the dropdown options menu $Delete$ with internal indexing 28, code selection with $LingusticVariable$ = $Delete\_select\_operace$. An example of the generated program code after editing by the developer is shown in Figure 8.



```
procedure TForm1.N28Click(Sender: TObject);
begin
if messageDlg('Delete an operation from the database?',mtConfirmation,[mbYes,mbNo],0)=mrYes
then Form1.IBDataSet_NAK_Operac.Delete;
//Form1.IBDataSet_NAK_Operac.Post;
Form1.IBDataSet_NAK_Operac.Close;
Form1.IBDataSet_NAK_Operac.Open;
DataModule_redaktor.IBStoredProc_update_detal_norma.ParamByName('detal_id').AsInteger := IBDataSet_NAK_DETAL.fieldByName('id_detal').AsInteger;
DataModule_redaktor.IBStoredProc_update_detal_norma.ExecProc;
IBDataSet_NAK_DETAL.Refresh;
IBDataSet_NAK_Operac.Refresh;
DBGridEh1.Refresh;
end;
```

Figure 8 – A fragment of the code after editing by the developer based on the method description 15.

Based on the examples given above, the developer is able to create and implement the CPPS cybernetic component with the help of the method description, and the possibility of program code "partial" generation. The completeness of program code generations directly depends on the DB content *“Container Solutions”*, that contains examples of event implementations (cod), which can be supplemented during the CPPS development work. This solution allows adapting the proposed method description to any object-oriented language, and also allows the developer to expand the DB with new *“Container Solutions”*, which will reduce the time at the programming stage in the future. The proposed solutions were implemented in the "Computer-aided design software for cyber-physical manufacturing systems" copyright certificate of Ukraine No. 74576 dated 09.11.2017.

3. CONCLUSION

As a result of the consistency of the modeling language syntactic constructions developed terminological basis and the terms in which the main structural components are described. In practice, it allowed us to develop an interpreter that is able to automatically translate the developed graphical interface (HMI), compiled in terminology close to a certain subset of natural language, its properties and properties of GUI elements, as well as events on the basis of which interaction with the user is implemented into the format of development environment commands and high-level language programming on which the CPPS cybernetic component is being developed. At the same time, the syntactic constructions of the developed language themselves are a script sequence and are quite simple for understanding by specialists and ordinary CPPS developers (examples 1-6). This allows us to assert that the use of the proposed modeling language by the developers makes it possible to reduce the development time of the CPPS cybernetic component at the early stages of drafting the technical requirement.

ACKNOWLEDGEMENTS

The author (Mohammad Ayaz Ahmad) would like to acknowledge the keen support in financial assistance for this work of the Vice Presidency / Studies and Scientific Research/Deanship of Scientific Research on behalf of University of Tabuk, Kingdom of Saudi Arabia and Ministry of Higher Education, K.S.A under the research grant no. S-0263-1436/dated 15-03-1436 [21]-[25], and also highly acknowledge the Department of Informatics, Kharkiv National University of Radio Electronics, Ukraine in numerous help and support to complete this article.

CONFLICT OF INTERESTS

The authors declare that there is no conflict of interests.

REFERENCES

[1] A.C. Pereira, F. Romero, A review of the meanings and the implications of the Industry 4.0 concept, Procedia Manuf. 13 (2017), 1206–1214.

[2] Y. Lu, Industry 4.0: A survey on technologies, applications and open research issues, J. Ind. Inform. Integr. 6 (2017), 1–10.

[3] P. Zheng, H. wang, Z. Sang, R.Y. Zhong, Y. Liu, C. Liu, K. Mubarok, S. Yu, X. Xu, Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives, Front. Mech. Eng. 13 (2018), 137–150.

[4] G. Matana, A. Simon, M.G. Filho, A. Helleno, Method to assess the adherence of internal logistics equipment to the concept of CPS for industry 4.0, Int. J. Product. Econ. 228 (2020), 107845.

[5] M. Sony, Industry 4.0 and lean management: a proposed integration model and research propositions, Product. Manuf. Res. 6 (2018), 416–432.

[6] C. Wittenberg, Human-CPS Interaction - requirements and human-machine interaction methods for the Industry 4.0, IFAC-PapersOnLine. 49 (2016), 420–425.

[7] P. Leal, R.N. Madeira, T. Romão, Model-Driven Framework for Human Machine Interaction Design in Industry 4.0, in: D. Lamas, F. Loizides, L. Nacke, H. Petrie, M. Winckler, P. Zaphiris (Eds.), Human-Computer Interaction – INTERACT 2019, Springer International Publishing, Cham, 2019: pp. 644–648.

[8] E. Lodgaard, S. Dransfeld, Organizational aspects for successful integration of human-machine interaction in the industry 4.0 era, Procedia CIRP. 88 (2020), 218–222.

[9] P. Fantini, M. Pinzone, M. Taisch, Placing the operator at the centre of Industry 4.0 design: Modelling and assessing human activities within cyber-physical systems, Computers Ind. Eng. 139 (2020), 105058.

[10] A.T. Jones, D. Romero, T. Wuest, Modeling agents as joint cognitive systems in smart manufacturing systems, Manuf. Lett. 17 (2018), 6-8.

[11] M.-P. Pacaux-Lemoine, Q. Berdal, S. Enjalbert, D. Trentesaux, Towards human-based industrial cyber-physical systems, in: 2018 IEEE Industrial Cyber-Physical Systems (ICPS), IEEE, St. Petersburg, 2018: pp. 615–620.

[12] E.A. Lee, Cyber Physical Systems: Design Challenges, in: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), IEEE, Orlando, FL, USA, 2008: pp. 363–369.

[13] A.J.C. Trappey, C.V. Trappey, U.H. Govindarajan, J.J. Sun, A.C. Chuang, A Review of Technology Standards and Patent Portfolios for Enabling Cyber-Physical Systems in Advanced Manufacturing, IEEE Access. 4 (2016) 7356–7382.

[14] E. Moshev, V. Meshalkin, M. Romashkin, Development of Models and Algorithms for Intellectual Support of Life Cycle of Chemical Production Equipment, in: A.G. Kravets, A.A. Bolshakov, M.V. Shcherbakov (Eds.),

Cyber-Physical Systems: Advances in Design & Modelling, Springer International Publishing, Cham, 2020: pp. 153–165.

[15] U.J. Tanik, Cyberphysical Design Automation Framework for Knowledge-based Engineering, J. Innov. Manage. 1 (2013), 158–178.

[16] E.O. Aliyu, O.S. Adewale, A.O. Adetunmbi, B.A. Ojokoh, Requirement Formalization for Model Checking using Extended Backus Naur Form, i-manager's J. Softw. Eng. 13(3) (2019), 1-6.

[17] A.F. Kurgaev, S.N. Grigoriev, Metalanguage of Normal Forms of Knowledge, Cybern Syst Anal. 52 (2016) 839–848. https://doi.org/10.1007/s10559-016-9885-3.

[18] M. Wang, L. Shen, Y. Deng, The Behavior of Mechanical Energy Storage Mechanisms and Representation, Adv. Sci. Lett. 4 (2011), 3077–3081.

[19] ISO/IEC 14977, Information technology – Syntactic metalanguage -- Extended BNF, 1996.

[20] M. Crepinsek, T. Kosar, M. Mernik, J. Cervelle, R. Forax, G. Roussel, On automata and language based grammar metrics, Computer Sci. Inform. Syst. 7 (2010), 309–329.

[21] S. K. Mustafa, M. A. Ahmad, V. Lyashenko, O. Zeleniy, Some Features of Route Planning as the Basis in a Mobile Robot, Int. J. Emerg. Trends Eng. Res. 8 (2020), 2074–2079.

[22] V. Lyashenko, S.K. Mustafa, N. Belova, M.A. Ahmad, Some Features in Calculation of Mold Details for Plastic Products, Int. J. Emerg. Trends Eng. Res. 7 (2019), 720–724.

[23] M. Ayaz, T. Sinelnikova, S. K. Mustafa, V. Lyashenko. Features of the Construction and Control of the Navigation System of a Mobile Robot, Int. J. Emerg. Trends Eng. Res. 8 (2020), 1445-1449.

[24] M. Ayaz, I. Tvoroshenko, J. H. Baker, V. Lyashenko. Computational Complexity of the Accessory Function Setting Mechanism in Fuzzy Intellectual Systems, Int. J. Adv. Trends Computer Sci. Eng. 8 (2019), 2370-2377.

[25] S.K. Mustafa, M.A. Ahmad, V. Baranova et al. Using Wavelet Analysis to Assess the Impact of COVID-19 on Changes in the Price of Basic Energy Resources, Int. J. Emerg. Trends Eng. Res. 8 (2020), 2907-2912.