https://www.sworldjournal.com/index.php/swj/article/view/swj09-01-009
DOI: 10.30888/2663-5712.2021-09-01-009

УДК 62.932:007.52

AUTOMATION OF FLEXIBLE HMI INTERFACE DEVELOPMENT FOR CYBER-PHYSICAL PRODUCTION SYSTEMS

Igor Nevliudov
EngD, Professor
ORCID: https://orcid.org/0000-0002-9837-2309
Vladyslav Yevsieiev
EngD, Professor
ORCID: https://orcid.org/0000-0002-2590-7085
Nikolaj Starodubcev
P.Hd., Asistant Professor
ORCID: https://orcid.org/0000-0001-7856-5771
Nataliia Demska
P.Hd., Asistant Professor
ORCID: https://orcid.org/0000-0002-9931-9964
Kharkiv National University of Radioelectronics, Kharkiv, Nauky Ave. 14, 61166

Abstract. *This publication is devoted to solving the problem of automation of flexible HMI interface development for monitoring and control of technological processes in cyber - physical production systems used in Smart Manufacturing within the Industry 4.0 concepts. The peculiarity of such systems is the great flexibility in use and upgrades, the minimum time of reconfiguration and implementation in the production process. As a result, there are questions about the implementation of adequate and modern HMI interface of the production process operator in real time, for timely analysis and decision making. Modern control systems for automated lines have a precise nature, each workbench has its own "control system", which is offered by the manufacturer, all these "control systems" are combined using the Industrial Internet of Things. However, such solutions are outdated, currently the one relevant is a single control system for a production line, shop, enterprise, corporation, which has a flexible interface that can be configured in a minimum of time, without the involvement of software developers. This is possible if the development of the HMI interface will be implemented on new approaches in the form of a specialized language based on natural language, which will reduce the time of development and implementation of additive cyber design.*

Keywords: *Industry 4.0; Industrial Internet of Things, Smart Manufacturing, Cyber-Physical Production Systems, Human Machine Interface, Graphical User Interface.*

Introduction

Global competition in the production of high-tech products is characterized by shortening life cycles (LC), complicating technical and technological preparation of production (TPP) and increasing requirements for their control and monitoring in real time [1-2]. Requirements for achieving high quality of such products require constant improvement of technological processes (TP), as well as changes in the structure of their management, which are central factors of success for manufacturing companies. With the development of Industrial Internet of Things (IIoT) and Industry 4.0, the introduction of Cyber-Physical Production Systems (CPPS), within the concepts of Smart Manufacturung, is becoming more widely used [3-5]. In turn, the digitization of production processes and management processes requires the processing and manipulation of large amounts of heterogeneous industrial data in real time and throughout the product life cycle. To use industrial data, to gain a competitive

advantage, it is necessary to provide flexible, educatable, Lean Production (LP), which must be human-centered [6].

But, despite the rapid development of research data, the development and implementation of CPPS still remains an individual task for each enterprise, and, as a result, there is no single approach to the management of organizational and technical production facilities.

To solve this problem, based on the proposed architectural and logical model of process control in complex organizational and technical production facilities and methods of its decomposition, the control process technology is developed [7]. On its basis the description system models that allow the developer to receive algorithms of functioning of organizational and technical production objects, both as a whole and at each stage are offered.

The Jump LC model is proposed and formalized, based on the synthesis of visual components and formal representation of properties and events, which allows to automate the development of additive cyber-design, based on the use of "solution containers" and "linguistic variables" [8].

It should be noted that the proposed decision difficult to understand users and is a mathematical core, for which it is necessary to develop formal language describing the structure and parameters of the developed additive cyber design that is as close and intuitive to some subset of natural language and applicable in a given subject matter area (SMA).

On the one hand, it was found that developers of additive cyber-design CPPS use binding to user interface elements and minimize the use of software code. This code, in most cases, is a handler of an event initiated by the user or internal processes, interacting with the necessary visual elements to solve a problem.

On the other hand - users who are specialists in SMA, can formulate a semantically correct structure and internal hierarchy of all interface elements with the necessary values of parameters and events that must be implemented to solve the tasks in the TOR.

There is a need to develop a syntactic and semantic model of a new formal language, building a clear and simple (for professionals and developers) additive cyber-design, close to some subset of natural language. This will help solve the problem of automatic transformation of semantic rules to describe the process of developing the HMI interface CPPS in a syntactically and terminologically correct structure, to implement its compilation in the required development environment.

1. Development of language model specification

Define the concept of language model (LM) - a declarative (non-procedural) language, the purpose of which is to define and describe the terminology, which is based on the proposed and the relationship between metadata and data of the subject area, as well as ways to transform them.

This paper proposes the following specification of the language of data models:

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

– keywords: basic concept in the form of words reserved in the developed

mathematical model (MM), which are used to describe key features such as:

Form – some selected and uniquely identified part of the subject area. Its purpose is to describe and present the visual structure of additive cyber design in the form of basic blocks;

$Form^{master}$ – main form for describing and presenting the visual structure of additive cyber design;

$Form^{slave}$ – an auxiliary form for presenting or entering visual information that is called from the main form;

ParameterForm – a set of types and methods of describing the properties of the subject area, selected and grouped by certain characteristics, as well as identified by name. Purpose - a description of the parameters necessary and sufficient to display and model a visual representation of $Form(Form^{master}, Form^{slave})$;

ElementForm – an element or group of GUI elements (subclass of an object) to visually represent the interaction between the user and the simulated functionality of the additive cyber design. They contain the necessary features for the implementation of the user interface, or management and interaction with information flows and data;

EvenForm – event or group of events (action) that may occur (have already occurred, or will occur) with the subject area, at a certain point or time interval. Identified by time (necessity) and the object to which the event belongs. With one object, at one time, only one event can occur, which is pre-initialized by the user;

ParametrElement – types and methods of describing the properties of elements, both individual and grouped by certain characteristics and identified by name. Purpose - a description of the parameters necessary and sufficient for the presentation and modeling of the visual representation of the element, within a single information object;

EventElement – an event, group of events, or condition that may occur (has already occurred or will occur) with a GUI that performs a specific function at a specific point or time interval. Identified by time (necessity) and the element to which the event belongs. Only one user-initiated event can occur with one object at a time. Purpose - one of the main properties of the element, which is limited: the functionality of this GUI element, scope (in terms of the need to use in this model of additive cyber-design CPPS), the need and role in the overall concept of application in the user interface for testing information flows;

ValueElement – value assigned to the type and method of describing the properties of the GUI element. Purpose - assignment of a specific value (integer, linguistic, Boolean) type, or method of describing the parameters, depending on the functional features and implementation of the visual intuitive interface of each part of the subject area in accordance with the algorithm. For some ParameterForm and ParameterElement in their functional purpose and nominal identification, the values may be the same, depending on the requirements of CPPS;

LingusticVariable – named (in the natural language of the system) logical description of actions in the event of events. Such descriptions can be grouped by a number of features. Purpose - assignment of an event class or a single event to a linguistically intuitive user-friendly model of the variable description of reactions in

the event of an event;

ContainerSolution – named description of reactions, when an event or group of events occurs, at a certain point in time on an element (group of elements), or subject area. Is rigidly structured, depending on the high-level programming language and development environment that is needed to achieve the development goal. Application - a partial or complete solution to perform the necessary actions with the data (information flows) necessary to achieve the goal of development, provided that the main goal of the development of additive cyber-design or at certain levels of decomposition;

parameter – the name of the parameter that describes or characterizes *Form* or the GUI element;

value – the value of the parameter, which can have both a digital value and a registered special word in the development environment;

name – *ContainerSolution* name, which is clear to the end user;

event – the name of the event that is possible on *Form* or the GUI element;

cod – part or fragment of program code that solves a problem and is stored in *ContainerSolution*.

➢ identifiers used to denote such features:

- an indication that parameters and events belong to domain or non-domain types: *domen*, *not_domen*. Domains of the corresponding characteristics (values) belonging to the listed (accounting) type which has a possibility of a choice from in advance formed list. An example is some *ParameterForm* parameters of *Form* display, which can take on values *true* or *false*. For example, the *Align* parameter, that is inherent in the $dom(parameter^{z}_{p\_список})$ and $dom(parameter^{h}_{p\_список})$, can take the values fixed by the CPPS development environment and specified in the TOR;
- data type values (*value*), which determines the characteristics of the parameters *ParameterForm* and *ParametrElement* (text, boolean, integer, integer negative, text, phrase);
- linguistic description of the reference feature *LingusticVariable* (*name*) on *ContainerSolution*, which contains the required *cod*;
- basic concepts of the Jump LC model, which make it possible to link events *ElementForm* and *EventElement*, contain a set of certain *event*, belonging to a certain visual graphic element with *ContainerSolution* (*cod*) through *LingusticVariable*(*name*).

As you can see, unlike keywords, the proposed identifiers can theoretically be redefined, but this leads to errors, so the above identifiers are included in a fixed dictionary of keywords.

➢ literals, a set of values that are not represented by an identifier.

String literals are represented as a sequence of allowed characters with different spellings (uppercase and lowercase) letters. For example, *name_form*, which is used in the parameters *Caption*, *Name* etc., as well as assigning a unique name (*name*)

for each *LingusticVariable*, which contains a certain piece of program code. For example, "save in the database", "calculate the result", etc., which are set by the end user for the convenience of the methodology being developed.

Algebraic letters – letters that are a description of simple logical operations of the type *True, False*, which allow to set values (*value*) of a parameter (*parameter*), belonging to *ParameterElement*, *ParameterForm* and is necessary and sufficient to describe the properties of the visual elements of the CPPS, or the function of the Collaborative Manufacturing Execution System (c-MES), in accordance with the necessary requirements.

Reserved letters are a word, phrase or abbreviation that allows you to select a property of the parameter required to achieve the conditions specified in the algorithm. An example is the *WindowsState* form property in the RadStudioXE16 development environment, to which you can select reserved abbreviations: *wsNormal,wsMinimized, wsMaximized*. That is, on first launch, the developer can specify the type of form display. *wsNormal* – default display, in the form in which it was created at the design stage, *wsMinimized* – the form that is displayed in a minimized form on the taskbar, *wsMaximized* – when launched, the form expands to the full size of the desktop.

Reserved letters can be common to *ParameterForm* and *ParametrElement*, as well as specialized, ie belong to a certain visual form, which determines the specifics of an element. However, it should be noted that the reserved letters to determine the values of a parameter of visual components that have the same purpose, can perform different specified functions and handle events in the same development environment.

Types of values presented, which contain some parameters of *ParameterForm* and *ParametrElement*, permissible in the field of application:

Integer data type (integer) – allows you to assign a parameter *ParameterForm* and/or *ParametrElement* a certain and necessary digital value of the dimension or coordinates of the location of the visual element relative to *Form*. Used mainly to describe visual graphic elements. It is the smallest logical element of a two-dimensional digital image in raster graphics (pixel). The length of the line depends on the screen resolution and the TOR requirements put forward by the customer to the developer.

Integer negative – allows you to assign a parameter a certain value that is within the range (-1,1,2,3,...,*n*), which belongs exclusively to *ParametrElement* and describes the numbering in this context:

-1 – numbering is missing, the parameter is not involved;

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

Textual / linguistic (char) – allows you to assign to the parameter a logical order of the values of the characters that contain the necessary user explanations or the name of the graphical elements required for ease of use with CPPS. Also, this type of value representation is used to specify a specific name *LingusticVariable*, assigned to the event *EvenForm*, *EventElement*.

Logical (boolean) – can only take two values: *true* or *false* and acts as a switch to use a parameter in *ParameterForm* and *ParametrElement*.

Text phrase (enumerated type) – the type of data specified by the list in the form of a domain, which allows you to specify a list of reserved words in the development environment or abbreviations that can take one or another *parameter* for *ParameterForm* and *ParametrElement*.

Dividers – symbolic designations of allocation of the basic elements of a syntactic construction developed LM.

<*Form*> (angle brackets *Form*) – are used to identify a keyword that indicates the beginning of a meta description of a particular *Form* in the design of LM.

</*Form*> (slash angle brackets *Form*) – is used to identify a keyword that indicates the completion of a meta description of a particular *Form* in the design of LM.

For the proposed keyword design, at the beginning and end of the meta description of *Form* the following restrictions are imposed: *Form* name can contain numbers like *Form*1, or a letter definition, for example, *Form_master* or *Form_add_operat*. In this case, the keyword of the beginning of the meta description must coincide with the keyword of the end of the meta description of a particular *Form* in the design of LM. If this design requirement is not met, the MM interpreter will not be able to perceive the content as a meta-description of all the necessary parameters and events inherent in the *Form*.

{ (opening bracket) – mandatory meta description at the beginning of the meta description line *Form* and *ElementForm*.

} (closing brackets) – mandatory meta description completion character *Form* and *ElementForm*.

(hash sign) – after this symbol, the design of the LM interpreter perceives the beginning of the description of the graphical visual elements of the user interface (*ElementForm*).

/# (slash hash sign) – after this combination of characters, the interpreter MM believes that the description of the graphical visual elements of the user interface (*ElementForm*) is completed.

/ (slash) – used to define hierarchies of meta-descriptions of visual graphic elements (*ElementForm*), according to the tree of construction of additive cyber design and is applied inside # /# meta description of *Form*. *ElementForm*1/*ElementForm*2 – must be understood as *ElementForm*2, that is inside *ElementForm*1 and is an integral part of it.

[ ] square brackets – used to meta-describe the required parameters and events *ParameterForm*, *EvenForm*, *ParametrElement*, *EventElement*.

; (semicolon) – mandatory LM construction symbol, which indicates that the assignment to this *parameter* or *event* corresponding *value* and *name* completed, applied internally.

, , (listing through a comma) – used to list *parameter* names for *ParameterForm*, *ParametrElement*, and also *event* for *EvenForm*, *EventElement* provided for a set of several *parameter* or *event* corresponding values *value* and

name are the same, applied internally.

= (equal sign) – assigns *parameter* a certain value *value* and is used to determine *event* certain *name* with *LinguisticVariable*, which contains a link to *cod* or its fragment in *ContainerSolution*. It should be noted that depending on the context (logic and content of the performed actions), this sign can be interpreted as an instruction of assignment, according to which for the specified basic parameter the value which belongs to it is defined.

Comments – all characters and strings written inside this construct by the LM interpreter are ignored and perceived as comments. Alphanumeric characters of national alphabets are allowed, supported by the operating system and development environment. The limitation for comments is that the sequence should not exceed 255 characters.

?** (question mark with two asterisks) – indicates that the specified characters will be followed by a comment that is ignored by the LM interpreter.

**? (two asterisks and a question mark) – shows that after the specified characters the comment ends and the text which is not ignored by the LM interpreter goes further.

To adapt the developed syntax of the description of LM, it is proposed to use the Bekus-Naur form. The rationale for this choice is that the extended Becus-Naur form is used to describe context-free grammars [9] and simplifies and reduces the scope of the description. The extended Becus-Naur form is described in the international standard [10]. The analysis of which showed that this form makes it possible to develop an intuitively simple and adaptive formal language for presenting and describing the necessary data for the development of CPPS, based on approaches to object-oriented programming.

Based on the above specification, the language of the data models and the basic concept of the Jump LC model, the following syntactic diagram of LM additive cyber-design is proposed. (fig. 1).

The syntactic diagram proposed in this study of the types of representation of values that may belong to the identifiers is presented in figure 2.

As you can see from Figure 2, the *Parameter* identifier, *Event* refers to *domen* (list) type and is represented as a text word or abbreviation related to *ParameterForm_name* and *EvenForm_name*, and *ParameterElement* and *EvenElement*, according to figure 1.

List of parameters (*parameter*) and events (e*vent*), belonging to *ParameterForm_name*, *EvenForm_name*, within a single development environment, is constant and unchanging. For a list of parameters (*parameter*) and events (e*vent*), belonging to *ParametrElement*, *EventElement* in accordance, the same limitation is that these visual graphic elements have the same purpose within the same development environment. It is worth noting that this type belongs to *value* for *ParametrElement* and *ParameterForm_name*, which contain a text word or abbreviation reserved by the development environment.

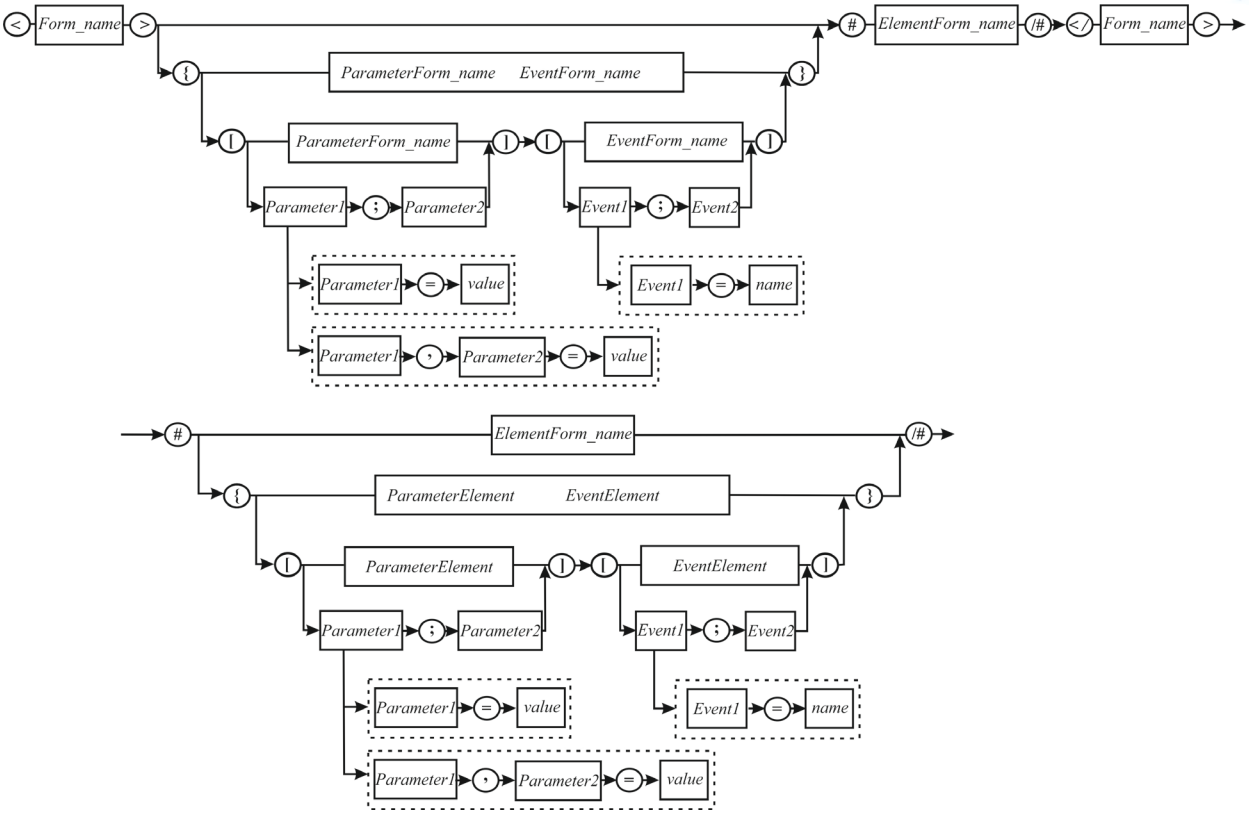


Fig. 1. Syntactic diagram of the modeling language

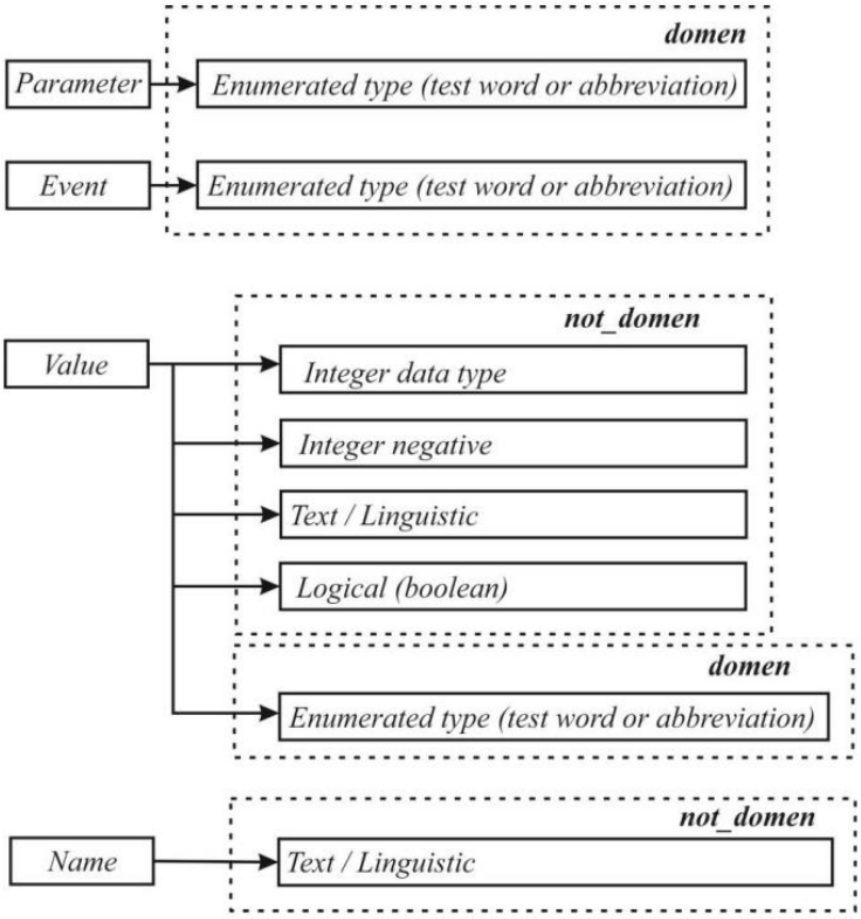


Fig. 2. Syntactic diagram of types of representation of values of identifiers

Identifiers *value* and *name* belong to *not_domen* (not accounting) type. This is justified by the fact that the value *value* can be set by the developer depending on the requirements of TOR for additive cyber design. For the *name* identifier, which is included in *LinguisticVariable*, the name that refers to *ContainerSolution*, contains the required fragment or part of the program code (*cod*). It is set by the user, taking into account its logical advantages and ease of use.

For the convenience of reading and presentation of the developed declarative language (Fig. 1-2) it is necessary that it has the quality of comprehension and reading. This can be achieved using at least three principles of language representation [11], namely:

– be as linear as possible;
– be brief;
– be self-documented.

Based on the proposed assumptions and recommendations, the following type of LM recording style is proposed for the developed CPPS declarative language, which allows to simplify and standardize the code..

Example 1

< Form _ master >

{ ?** opening a block describing parameters and values, as well as events and names *LinguisticVariable* для *Form _ master* **?

[*parameter*1 = *value*; *parameter*2, *parameter*3 = *value*]

[*event*1 = *name*; *event*2 = *name*]

} ?** closing the block describing parameters and values, as well as events and names *LinguisticVariable* для *Form _ master* **?

" the name of the item in the development environment"

?** opening the block of description of visual graphic elements *Form_master* **?

{ ?** block of *Element*1_*Form_master* description **?

[*parameter*1 = *value*; *parameter*2, *parameter*3 = *value*]

[*event*1 = *name*; *event*2 = *name*]

} ?** closing the block of *Element*1_*Form_master* description **?

{ ?** block of *Element*2_*Form_master* description **?

[*parameter*1 = *value*; *parameter*2, *parameter*3 = *value*]

[*event*1 = *name*; *event*2 = *name*]

} ?** closing the block of *Element*2_*Form_master* description **?

/# ?** closing the block of description of visual graphic elements of *Form_master* **?

< / Form _ master >

If necessary, implement a hierarchy (construction tree) of *ElementForm*1/*ElementForm*2 visual graphic elements the following fragment of the meta-description structure is offered:

Example 2.

“ the name of the item in the development environment” ?** opening the block of description of *Form_master* visual graphic elements **?

{ ?** block of *Element*1*Form _ master* description **?

[*parameter*1 = *value*; *parameter*2, *parameter*3 = *value*]

[*event*1 = *name*; *event*2 = *name*]

} ?** closing the block of *Element*1*Form _ master* description **?

/ “ the name of the item in the development environment”

{ ?** block of *Element*2*Form _ master* description **?

[*parameter*1 = *value*; *parameter*2, *parameter*3 = *value*]

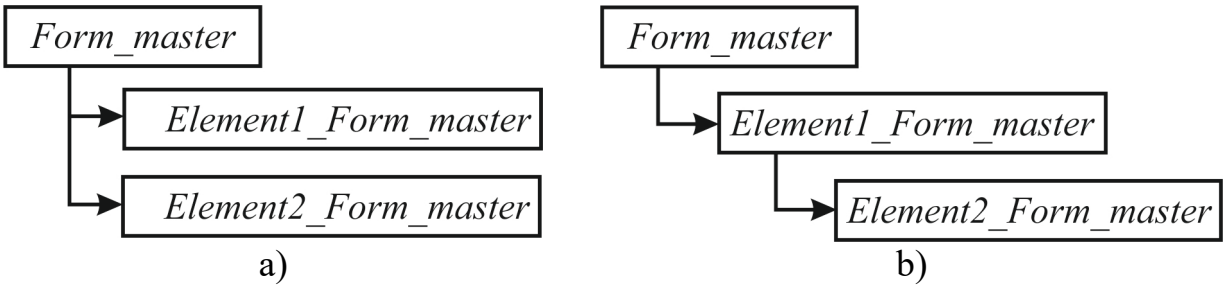
[*event*1 = *name*; *event*2 = *name*]

} ?** closing the block of *Element*2*Form _ master* description **?

/# ?** closing the block of description of *Form_master* visual graphic elements **?

Using “/” (slash) will allow the LM interpreter to determine the degree of nesting (belonging) of the visual element in another, ie the ability to implement a structure tree (Structure) of additive cyber-design in the development environment. Figure 3 shows a graphical representation of the structure of the construction tree *Form _ master* construction tree of CPPS (example 1, and example 2, b).

To determine the appropriate *value* and *name*, in the examples above, in *LingusticVariable* for *parameter* and *event* accordingly, after (=) the value type is set. In the absence of values, or the use of values reserved by the default development environment, this parameter is not declared in the meta description (not indicated).



a) *Element*1*Form _ master* і *Element*2*Form _ master* definitely belongs *Form_master*;

b) *Element*2*Form _ master* belongs *Element*1*Form _ master*;

Fig. 3. Graphical representation of the CPPS structure tree

Example 3 shows a meta-description of creating a blank form in the environment RadStudio XE6 for VLC Form Application.

Based on the meta-description given in (1), a graphical representation of the simplest user form was generated (Fig. 4).

A fragment of the meta description of additional visual graphic elements of the *Standard-Button* type (a custom button that performs a specific event) is presented in (2).

Example 3

$$
\begin{aligned}
&<Form1> \\
&\quad \{ \\
&\qquad [Caption = example\ 1, ClientHeight = 464, ClientWidth = 687, \\
&\qquad Height = 503, Name = Form\_master, Width = 703] \\
&\quad \} \\
&</Form1>
\end{aligned} \tag{1}
$$

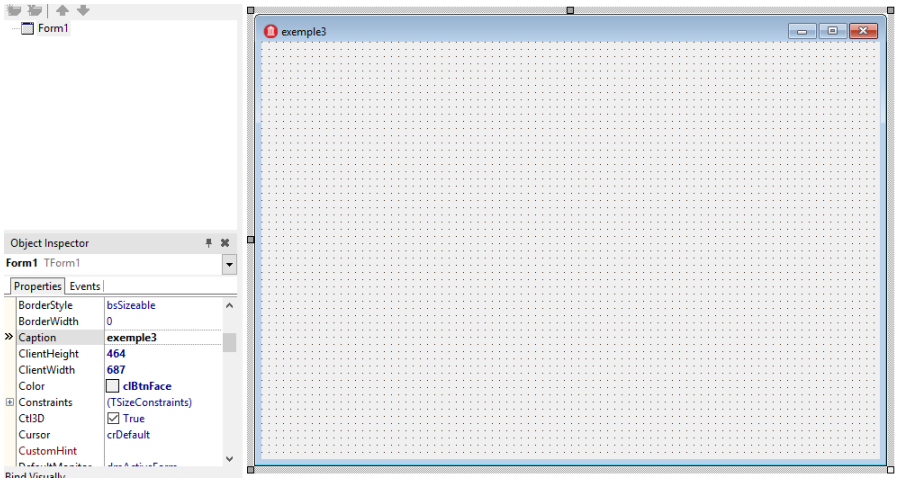


Fig. 4. A fragment of the RadStudio XE6 development environment of the simplest user form

$$
\begin{aligned}
&\#\ \text{“Buttion\_close”} \\
&\qquad [Caption = Close, Height = 33, Top = 408, Left = 560, \\
&\qquad Name = Buttion\_close, Width = 91] \\
&/\#
\end{aligned} \tag{2}
$$

Figure 5 shows an example implementation of a form with a Button element, the meta-description of which is given in (1) and (2), respectively.

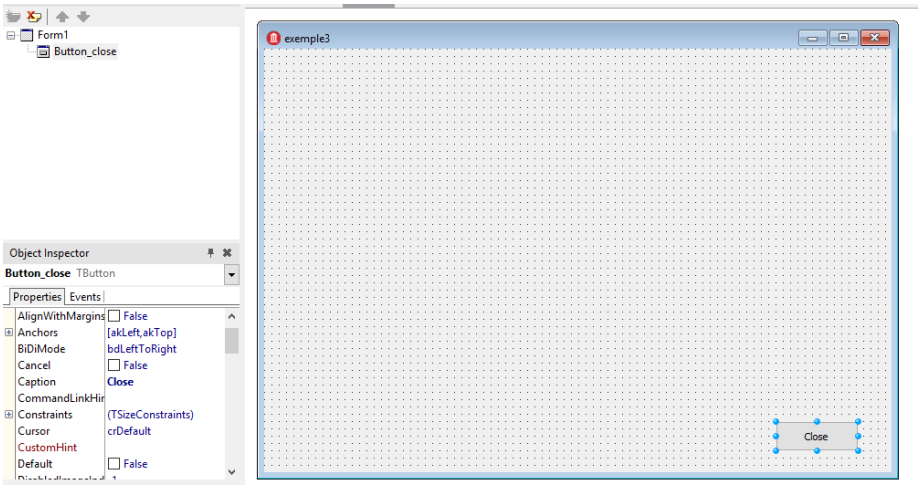


Fig. 5. A fragment of the development environment with the implementation of the form and the graphic element Button

In addition to the implementation of the graphical visual interface shown in fig. 4–5, based on meta-descriptions (1) - (2) the program code in Pascal language was generated, presented in fig. 6.

```
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.
```

Fig. 6. Program code in Pascal

Each element of the LM description given in the syntactic model (1) - (2) is written according to the syntactic diagram (fig. 1) and the diagram of types of representation of values of identifiers (fig. 2). The semantic model of LM is a system of values attributed to constructions and developed syntactic LM (interpretations of construction). This model is presented in the process of interpretation (analysis) of the proposed rules of description and presentation of the LM specification, symbols and their combinations.

Consider the meta-description of example (2), to determine the need to implement the attachments (accessories) of one visual element in another, as shown in figure 3, b. According to the proposed syntactic model (fig. 1), the meta-description will take the following form:

$$
\begin{aligned}
&\#\ \text{“}GrupBox1\text{”} \\
&\quad [Caption = GroupBox1, Height = 186, Top = 272, \\
&\quad Left = 4, Name = GroupBox1, Width = 678] \\
&\quad /\ \text{“}Button1\text{”} \\
&\quad [Caption = Close, Height = 32, Top = 146, Left = 585, \\
&\quad Name = Buttion1, Width = 86] \\
&/\#
\end{aligned}
\tag{3}
$$

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" $Form1$, based on which the user interface is developed, according to the TOR on CPPS and algorithm of functioning. Figure 7 shows a fragment of the generated user interface, according to the meta description (3) in the development environment RadStudio XE6.

From the above fragments of meta-descriptions (1) - (3) and fig. 3 you can specify any depth of embedding of graphical elements of the user interface. This

makes it possible to implement, using the proposed syntactic diagram of LM (fig. 1), the structure of CPPS of any degree of complexity and thus simplify the process of developing a visual component based on an object-oriented approach to programming.

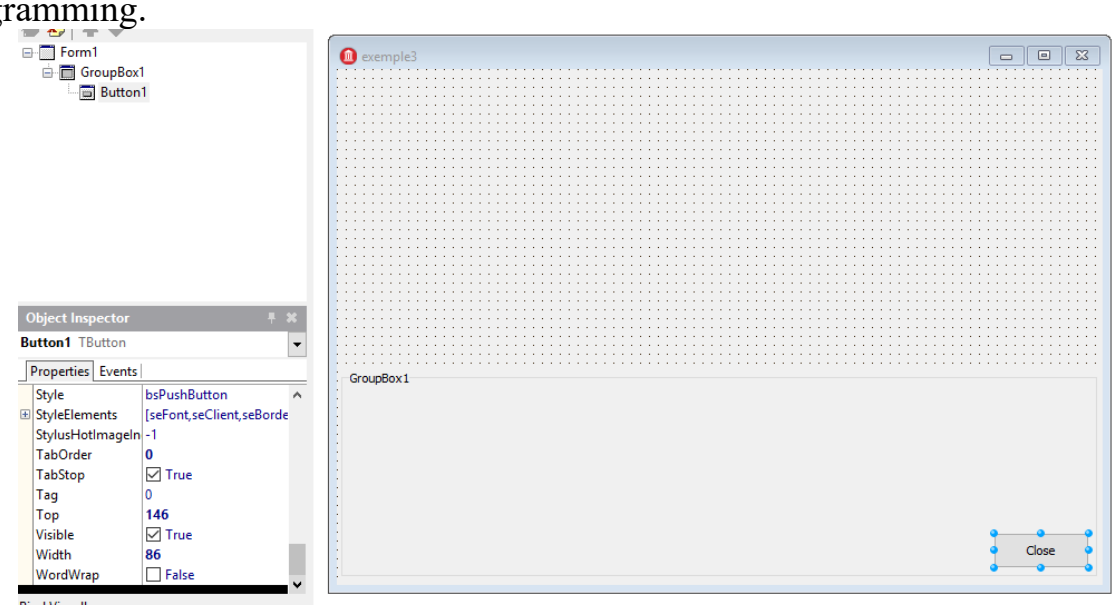


Fig. 7. Implementation of the interface *Form*1

2. Development of syntax of meta description of events

Based on the proposed syntactic diagram presented in Figure 1, the following meta-description of events (*event*) for *Form* and *ElementForm* is proposed. Based on (1), we give an example of a simple meta-description of operation on an element *Buttion* event *Onclick*, to practice *LingusticVariable* with the name *Close_All*.

Example 4

$$
\begin{aligned}
&< Form1 > \\
&\quad \{ \\
&\quad [Caption = example\ 1, ClientHeight = 464, ClientWidth = 687, \\
&\quad Height = 503, Name = Form\_master, Width = 703] \\
&\quad\quad [Caption = Close, Height = 33, Top = 408, Left = 560, \\
&\quad\quad Name = Buttion\_close, Width = 91, OnClick = Close\_All] \\
&\quad \} \\
&< /Form1 >
\end{aligned}
\tag{4}
$$

As can be seen from the meta description in example (4), the developer describes the existence of an event on the element *Buttion* with the name $Name = Buttion\_close$, *LingusticVariable* under the name $Close\_All$ for the event *Onclick*. This representation allows the developer to implement a model of the LC process of managing the development of CPPS, which is presented in section 4 in the form of a sequence of links:

$$event \rightarrow LingusticVariable \rightarrow ContainerSolution \rightarrow cod \tag{5}$$

Based on the results of meta-description (4), the developer receives the generated program code, shown in figure 8.

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

Fig. 8. The result of generating program code

Consider a fragment - an example of a meta-description of example (6) implementation of a more complex code design, which was generated during the development of "System of operational dispatching management of the production enterprise" OSCEM "[12].

On *Form*1 there is an element *DBGrid _operac* to display information from the database (DB). In *Properties_DBGrid _operace* the binding to the visual drop-down element of the interface *PopupMenu_operac* is specified. It is necessary to generate the program code of removal from a DB of the selected record in *DBGrid _operac* .

Example 5

$$
\begin{aligned}
&< Form1 >\\
&\quad \vdots\\
&\quad \#\ \text{“}DBGrid\_operace\text{”}\\
&\quad \{\\
&\qquad [DataSurce = Form1.IBDataSet\_Nak\_Operac,\\
&\qquad Height = 120, Left = 344, Top = 24, Width = 320,\\
&\qquad Name = DBGrid\_operace,\\
&\qquad PopupMenu = PopupMenu\_operac]\\
&\quad \}\\
&\qquad \vdots\\
&\quad \#\ \text{“}DBGrid\_operace\text{”}\\
&\quad \{\\
&\quad [Name = PopupMenu\_operac, Items = 28,]\\
&\quad [OnClick = N28Click]\\
&\quad \}\\
&\quad \vdots\\
&\quad \#\ \text{“}N28\text{”}\\
&\quad \{\\
&\quad [Caption = Delete, Name = 28]\\
&\quad [OnClick = Delete\_select\_operace]\\
&\quad \}\\
&\quad \vdots\\
&< / Form1 >
\end{aligned}
$$

An example of the generated program code to implement the event per element *PopupMenu _ operac*, when pressed once, in the drop-down menu of options *Delete* with internal indexing 28, the code selection from *LingusticVariable = Delete _ select _ operace*. An example of the generated program code after editing by the developer is presented in fig. 9.

```
procedure TForm1.N28Click(Sender: TObject);
begin
if messageDlg('Удалить операцию из базы данных?',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;
```

Fig. 9. A fragment of the code after editing by the developer

Based on the examples above, the developer gets the opportunity, based on the proposed model and methods of CPPS development [7], as well as on the basis of the developed model of LF [8], using meta-description to create and implement additive cyber-design, with the possibility of "partial" generation of program code [13]. The completeness of the generation of program code directly depends on the content of DB "*Container Solutions*", which contains examples of events (*cod*) that can be performed in the process of developing additive cyber-design. This solution allows you to adapt the proposed meta-description to any object-oriented language, and also allows the developer to expand the database with new "*Container Solutions*", which will reduce time spent in the development of additive cyber design for process management in complex organizational and technical production facilities.

Conclusions

The complexity of developing the cyber component of CPPS is a complex scientific and applied task, due to the fact that the HMI setup must be carried out in parallel with the installation and commissioning of production lines, so the development of HMI control and monitoring system must be timely. software development is not cost effective. This necessitated the development of a new declarative language for the definition and manipulation of data, close to some subset of natural language. Therefore, the authors of this article proposed such solutions:

1. The specification of language of data models (keywords, identifier, literals, types of the presented values, etc.) is developed.

2. The syntactic diagram of the developed language which gives the chance to describe parameters, properties, values, and also events inherent both *WindowsForm* and GUI elements of HMI of the user interface of additive cyber-design is offered.

3. The syntactic diagram of types of representation of values of identifiers is developed.

4. Based on the results of the consistency of the developed terminological basis of syntactic constructions of LM and terms, an interpreter has been developed that is able to automatically translate HMI compiled in terminology close to some subset into a natural development command format and a high-level programming language. The syntactic constructions of the developed language are a script sequence and are

quite simple to understand for both specialists and ordinary developers (examples 1-5). This gives grounds to claim that the use of the proposed LM by the developers makes it possible to reduce the development time of additive cyber-design process control in complex organizational and technical production facilities based on CPPS.

REFERENCES:

1. Frank, A. G., Dalenogare, L. S., Ayala, N. F. (2019). Industry 4.0 technologies: Implementation patterns in manufacturing companies. International Journal of Production Economics, Vol. 210(C), P. 15–26, DOI:10.1016/ j.ijpe.2019.01.004.

2. Haseeb, M., Hussain, H. I., Ślusarczyk, B., Jermsittiparsert, K. (2019). Industry 4.0: A solution towards technology challenges of sustainable business performance. Social Sciences, Vol. 8(5), P. 154, DOI:10.3390/ socsci8050154.

3. Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. Journal of industrial information integration, Vol. 6, P. 1-10, DOI:10.1016/j.jii.2017.04.005.

4. Haleem, A., & Javaid, M. (2019). Additive manufacturing applications in industry 4.0: a review. Journal of Industrial Integration and Management, Vol. 4(04), P. 1930001, DOI:10.1142/S2424862219300011.

5. Yevsieiev V., Bronnikov A. Information systems development methodologies application analysis for cyber-physical production systems development. III International scientific-practical conference “Theory, science and practice” (Japan, Tokyo, 5–8 October 2020). P. 398–401. DOI: 10.46299/ISG.2020.II.III.

6. Yevsieiev V., Bronnikov A. Analysis of the cyber-physical production systems implementation impact to achieve the goals of lean production. The IIth International scientific and practical conference «Development of scientific and practical approaches in the era of globalization» (USA, Boston, 28–30 September. 2020). P.221–226. DOI:10.46299/ISG.2020.II.II.

7. Nevliudov, I., Yevsieiev, V., Maksymova, S., Filippenko, I. (2020), "Development of an architectural-logical model to automate the management of the process of creating complex cyber-physical industrial systems", Eastern-European Journal of Enterprise Technologies, Vol. 4, No. 3 (106). P. 44–52. DOI: 10.15587/1729-4061.2020.210761

8. Nevlyudov I., Yevsieiev V., Miliutina S., Kollesnyk K. Object semantic model for life cycle model “Jamp”. CAD in Machinery Design. Implementation and Educational Issues. 25 Proceedings of Polish- Ukrainian Conference (CADMD’2017). (Polish, Bielsko Biala, 20–21 October 2017). P. 31–32.

9. Plotnikova Z.V., Evseev V.V. (2009). Metod niskhodyashchego analiza s prognoziruemym vyborom al'ternativ dlya konstekstno – zavisimykh gramatik. Vostochno-evropeiskii zhurnal peredovykh tekhnologii. Vyp. 4/11(40). S.11–13.

10. The structure of the administration shell: trilateral perspective from France, Italy and Germany. [Electronic resource]. URL: https://www.de.digital/DIGITAL/Redaktion/EN/Publikation/the-structure-of-the-administration-shell.pdf?__blob=publicationFile&v=3. (Date of application: 14.09.2021).

11. V.A. Morozova, V.I. Pautov.(2017). Predstavlenie znanii v ekspertnykh sistemakh. Ekaterenburg.Izd.-vo Ural. Un-ta.. S – 120. ISBN 978-5-7996-2037-0.

12. Nevliudov,I., Yevsieiev, V., Demska,N., Novoselov, S. Development of a software module for operational dispatch control of production based on cyber-physical control systems. Innovative Technologies and Scientific Solutions for Industries. 2020. No.4(14), P.155–168. DOI:10.30837/ITSSI.2020.14.155.

13. Nevliudov I., Yevsieiev V., J. H. Baker, Ahmad M. A., Lyashenko V. (2021). Development of a cyber design modeling declarative language for cyber physical production systems. Journal of Mathematical and Computational Science. No.1. P.520–542, DOI:10.28919/jmcs/5152.

Анотація. *Дана публікація присвячена вирішенню питання автоматизації розробки гнучкого HMI інтерфейсу для моніторингу та керування технологічними процесами в кібер-фізичних виробничих системах, які використовуються в Smart Manufacturing, в рамках концепції Industry 4.0. Особливістю таких систем є велика гнучкість при використанні та модернізації, мінімальний час переналадки та впровадження у процес виробництва. Тому виникають питання реалізації адекватних та сучасних HMI інтерфейсів оператора виробничого процесу, в режими реального часу, для своєчасного аналізу та прийняття рішень. Сучасні системи керування автоматизованими лініями мають індивідуальний характер, на кожному верстаті є своя «система керування», запропонована виробником, всі ці «системи керування» об'єднуються за допомогою Industrial Internet of Things. Однак, такі рішення є застарілими. На даний час актуальним є рішення створення єдиної системи керування виробничою лінією, цехом, підприємством, корпорацією, яка має гнучкий інтерфейс, що легко налаштовується в мінімальний час без залучення розробників програмного забезпечення. Це можливо, якщо розробка HMI інтерфейсу, буде реалізоване на нових підходах у вигляді спеціалізованої мові, на базі природної мови, що дозволить скоротити час розробки та впровадження адитивного кібер-дизайну.*

Keywords: *Індустрія 4.0; Промисловий Інтернет Речей, Розумне Виробництво, Кібер-Фізичні Виробничі Системи, Людино-Машинний Інтерфейс, Графічний Інтерфейс Користувача.*

Article sent: 16.09.2021 г.