ІННОВАЦІЇ

UDC 001.895:330.341.1

V. I. Chubaievskyi, Candidate of Political Sciences,
K. O. Palahuta, Candidate of Economic Sciences, Associate Professor,
A. M. Desiatko, Senior Lecturer

TECHNOLOGIES OF MULTILEVEL STRUCTURES MODELLING ON THE EXAMPLE OF THE PROBLEM OF COMPLETING PRODUCTS

Urgency of the research. *One of the typical tasks encountered in the de-signing of intelligent systems is modelling of the multilevel structures for solving various applied problems.*

Target setting. *Consideration the possibilities of the language of artificial intelligence Visual Prolog for the implementation of recursive technology-based on the example of solving a multi-level task of product configuration.*

Actual scientific researches and issues analysis. *Such scientists as Biletsky O. B., Lytvyn V. V., Chery S., Gottlob G., Luger G. F., Russell G.F. made significant contribution to the development of the theory, methodology of artificial intelligence application for solving problems in the field of economics.*

Uninvestigated parts of general matters defining. *At the same time, insufficient scientific works highlight the features of the introduction of modern means of artificial intelligence for solving multilevel economic problems.*

The research objective. *Analyze existing approaches to solving multi-level tasks. To propose an effective tool for solving multilevel tasks using artificial intelligence language Visual Prolog.*

The statement of basic materials. *The problem of modeling of multilevel structures in intellectual systems on the basis of iterative and recursive technologies on the example of the problem of components is considered. The main characteristics of such structures are presented, their complexity is determined and the necessity of finding effective methods for their presentation and processing in the memory of the machine is given. There are two important paradigms in the development of recursive technologies: functional and logical programming. We consider the corresponding languages of artificial intelligence: Lisp and Prolog, their heirs and the most powerful language of Visual Prolog. The classical well-known iterative algorithm and the recursive program on the Prologue of solving the problem of the components of the internal combustion engine, as well as the recursive program on Visual Prolog, developed by the authors of the article, are given. Their comparison is made from the position of using the number of structures for the presentation of the information base, the cost of memory for their preservation, the complexity of developing and debugging the program, the ease of perception of their work.*

Conclusions. *The power of the language Visual Prolog is emphasized, which is especially manifested in the tasks of processing multi-level structures.*

Keywords: *recursive technologies; Visual Prolog language; multilevel structures.*

DOI: *10.25140/2410-9576-2019-2(18)-6-14*

УДК 001.895:330.341.1

В. І. Чубаєвський, к. п. н.,
К. О. Палагута, к. е. н.,
доцент,
А. М. Десятко, ст. викладач

ТЕХНОЛОГІЇ МОДЕЛЮВАННЯ БАГАТОРІВНЕВИХ СТРУКТУР НА ПРИКЛАДІ ЗАДАЧІ КОМПЛЕКТАЦІЇ ПРОДУКЦІЇ

Актуальність дослідження. *Одним з типових завдань, що виникають у процесі деформування інтелектуальних систем, є моделювання багаторівневих структур для вирішення різних прикладних задач.*

Постановка проблеми. *Розгляд можливостей мови штучного інтелекту Visual Prolog для реалізації рекурсивної технології на основі прикладу вирішення багаторівневої задачі комплектації продукції.*

Аналіз останніх досліджень і публікацій. *Значний внесок у розвиток теорії, методології застосування штучного інтелекту для вирішення задач у сфері економіки внесли такі вчені, як Білецький О. Б., Литвин В. В., Чері С., Готтлоб Г., Лугер Г. Ф., Рассел С. Дж. та ін.*

Виділення недосліджених частин загальної проблеми. *У той же час недостатньо наукових праць висвітлюють особливості впровадження сучасних засобів штучного інтелекту для вирішення багаторівневих економічних задач.*

Постановка завдання. *Проаналізувати існуючі підходи до вирішення багаторівневих завдань. Запровадити ефективний інструмент для вирішення багаторівневих завдань за допомогою мови штучного інтелекту Visual Prolog.*

Виклад основного матеріалу. *Розглянуто проблему моделювання багаторівневих структур в інтелектуальних системах на основі ітераційних і рекурсивних технологій на прикладі задачі компонентів продукції. Наведено основні характеристики таких структур, визначено їх складність та необхідність пошуку ефективних методів їх подання та обробки в пам'яті комп'ютера. У розвитку рекурсивних технологій є дві важливі парадигми: функціональне та логічне програмування. Розглядаються відповідні мови штучного інтелекту: Lisp і Prolog, їх спадкоємця - найпотужніша мова Visual Prolog. Наведено класичний відомий ітераційний алгоритм і рекурсивна програма на мові Prolog розв'язання задачі компонентів двигуна внутрішнього згоряння, а також рекурсивна програма на Visual Prolog, розроблена авторами статті. Їх порівняння зроблено з позиції використання кількості структур для представлення інформаційної бази, вартості пам'яті для їх збереження, складності розробки і налагодження програми, простоти сприйняття роботи.*

Висновки. *Підкреслюється потужність мови Visual Prolog, яка особливо проявляється в задачах обробки багаторівневих структур.*

Ключові слова: *рекурсивні технології; мова програмування Visual Prolog; багаторівневі структури.*

ІННОВАЦІЇ

Urgency of the research. One of the typical tasks encountered in the designing of intelligent systems is modelling of the multilevel structures for solving various applied problems. Such structures are characterized by the variable amount of levels in the hierarchy and the amount of elements in every level depending on one or another object of the calculation; by a repeatability of the procedures which are carried out in the processing of the elements of every level of this hierarchy; by their significant size and complexity of the implementation of the procedures that leads to the considerable spending of resources and, as a result, to the searching for the effective methods of their presentation and processing in the machine memory.

Calculation tasks based on such structures can be solved by implementing traditional branched and cyclical processes using the appropriate standard programming language operators (iterative programs). But involving of the recursive procedures in such calculations significantly changes the computing technology which is typical for the methods of artificial intelligence and ultimately can cause the additional positive results, such as simplification and universality of the algorithms.

Actual scientific researches and issues analysis. A lot of workings and publications are devoted to the recursive computing technologies and programming [1; 3; 4; 5; 7; 8; 9; 10]. Historically, these technologies have been developing on the basis of two important paradigms – the functional programming and the logical one. The functional programming began with the creation and realization of the Lisp language, and the logical one – with the Prolog language. An important feature of these languages is the focus not on the numerical processing, but on the character one. Therefore, data structures, which are supported by these languages, are not limited by the arrays. Lists are the basic structure of data which are maintained both by the Lisp language and by the Prolog one [3].

The presentation of data and programs in the Lisp language in the same form allows, if it is necessary, to interpret data as a program. It gives the opportunity for some Lisp programs to generate other programs, that characterizes the Lisp language as a flexible one. The programming model, involved in the Lisp language, had proved to be so good, that many other languages, including Erlang, APL, ML, Scala (multiparadigmal language), Haskell, and so on, were established on the principles of the functional programming.

Prolog is the most famous example of the logical programming language. It is characterized by a higher level of abstraction. The Lisp language allows a programmer not to care about the dynamic allocation of the memory for the lists, but the Prolog language, in addition, contains the built-in mechanisms of management of the program fulfillment. Because of that, the Prolog program, to a greater extent, is the declarative description of the subject area relations. Implementation of the programs implies the processes of setting the questions from the subject area and the automatic searching for the answers with the help of the built-in mechanisms of the logical output. Such approach is possible because Prolog originates from the logic of predicates. Prolog is based on Horn clauses (a subset of first-order logic) [3]. There are different versions of the Prolog language, such as Turbo-Prolog, GNU Prolog and other. A constantly updated overview of the Prolog realizations can be found at: http://dmoz.org/Computers/Programming/Languages/Prolog/ Impementations [4]. To the list of the latest most interesting products it should be added Visual Prolog, the possibilities of which will be further involved in this work.

The Prolog language does not contain the direct way of carrying out the repetition with the help of such constructions as FOR, WHILE, REPEAT, which are used in the Pascal, Basic or C languages. Prolog provides only two types of repetitions. There are rollback and recursion. Recursion is the process in which a function calls itself. Recursion has three main advantages: it can express the algorithms, which cannot be comfortably expressed by any other means; in the logical meaning it is simpler than iteration; it is widely used in the lists processing. There are few forms of recursion, such as simple recursion, parallel branching of recursion, mutual recursion, recursion of a higher order. In Prolog, tail recursion is often used and optimized in order to minimize the memory consumption on its implementation.

Let us further consider the problem of modeling multilevel structures on the example of solving the problem of breaking up the structure of complex products, also sometimes defined as the problem of a

product mix. The essence of this task is to determine the quantitative inclusion of components and elements in the product, has a multilevel structure.

The complexity of this task allows to try various automatized solutions: approaches, methods and programming technologies. In the case of implementation of these solutions by a lot of methods, later it is possible to conduct a comparative analysis and define the abilities of the applied approaches.

Consider the possibilities of the language of artificial intelligence Visual Prolog for the implementation of recursive technology-based on the example of solving a multilevel task of product configuration.

The research objective.

1. Analyze existing approaches to solving multilevel tasks.

2. To propose an effective tool for solving multilevel tasks using artificial intelligence language Visual Prolog.

The statement of basic materials. Let us consider the solution of the component problem based on the one hand application of simple procedures using procedural and object-oriented language technologies (Pascal, C, C ++, etc.) And on the other hand, Запропонувати

We will base on two approaches, which have the features of two relevant methodologies and are historically famous:

- solving the task of the building component products, proposed by O. B. Biletsky and V. S. Mykhailov [2];
- solving the component task by the Prolog language, proposed by S. Ceri, G. Gottlob, L. Tanka [10].

Note that this is the same task, although they are called differently and relate to different subject areas.

We will consider the proposed solutions on the example of the car engine components (Fig. 1), which is given in the book by S. Ceri [10].

The main property of complex products is the multilevel occurrence of simple elements in more complex ones. The mathematical analogue of such a structure is a finite graph without contours. A graph (G, X) is a collection of (X) elements — vertices and their relations — arcs (G).

The vertices of the graph correspond to the elements of the structure of the finished product, and the arcs correspond to the occurrence of lower level elements in the above. The weight of the i-th arc qi quantitatively characterizes the occurrences. Arcs are directed from products to their elements.

A vertex into which no arc enters is called a root. Vertices from which no arc extends are called leaves. The graph of the structure of production can be, for example, the design of a building, an airplane, a bridge, a power station, etc. The sets of the graph can be divided into tiers. In this set it is possible to distinguish nodes, which are called parents, and his subordinates - brothers, and each of them is the son of the father of the family [2].

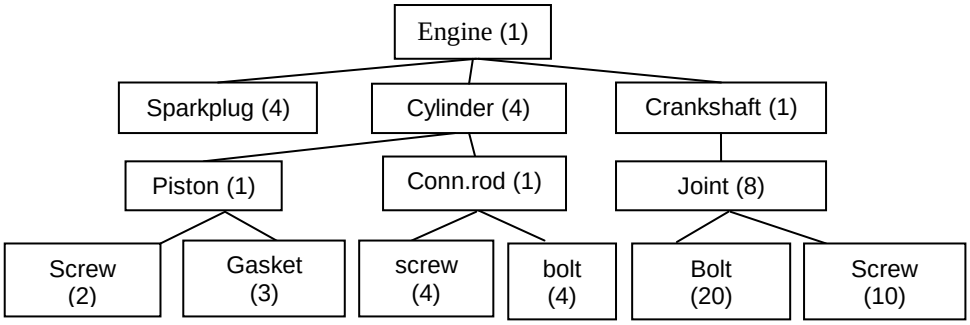

Fig. 1 The components of the internal combustion engine

To calculate the entry of components and elements into products, it is proposed in [2] to create 4 tables “Plan”, “Catalog”, “Input” and “Norm”. They implement an algorithm. We form them in accordance with Fig. 1.

ІННОВАЦІЇ

Table 1 **- Table "Plan"**

Father identifier	Amount
Engine	1

Table 2 **- Table "Catalogue"**

Father		Son address
№ of the tier	Identifier	1
1	Engine	1
1	Engine	2
2	Crankshaft	6
3	Piston	7
3	Conn.rod	9
3	joint	11

Table 3 **- Table "Input"**

№ of the area	Son identifier	Input	Brother address
1	Sparkplug	1	2
2	Cylinder	4	3
3	Crankshaft	1	1
4	Piston	1	5
5	Conn.rod	1	4
6	joint	8	6
7	screw	2	8
8	gasket	3	7
9	screw	4	10
10	bolt	4	9
11	screw	10	12
12	bolt	20	11

Table 4 **- Table "Norm"**

Father identifier	Amount
Sparkplug	1
Cylinder	4
Crankshaft	1

The explosion algorithm is performed in several stages.

At the first stage, data shown in the graph (Fig. 1) are placed according to the spreadsheets "Plan", "Catalogue" and "Input".

At the second stage, the ith record of the jth tier is read from the catalogue, and sth address of a bigger brother is determined from this record.

At the third step, the transition to the "Input" table to the sth record of the bigger brother is carried out, the input is defined and after that is multiplied on the product value in the "Plan" table. The result should be written in the "Norm" table.

At the fourth stage, the transition to the brothers records is fulfilled, and the same to the third stage actions are performed.

At the fifth step, formed data in the "Norm" table are sorted, and, if it is necessary, summing up the values of requisites with the same identifiers is implemented.

At the sixth step, data of the "Norm" table is transferred to the "Plan" table. Then, actions in accordance with the 1-5 steps are carried out.

To implement this algorithm, it is enough to use simple procedures with the variables and arrays determination, and to use the assignment, branching, and cycles operators.

The second approach to solve this problem contains a significantly simpler information base, which consists of two tables. At the same time, the task is slightly expanding. Using the second table makes it possible to calculate the total cost of the components. We give them in the form of the original.

The second approach to solve this problem contains a significantly simpler information base, which consists of two tables. At the same time, the task is slightly expanding. Using the second table makes it possible to calculate the total cost of the components. We give them in the form of the original.

Table 5 **- Table "COMPONENT"**

Engine	Sparkplug	4	b
Engine	Cylinder	4	c
Engine	Crankshaft	1	c
Cylinder	Piston	1	c
Cylinder	Conn.rod	1	c
Crankshaft	Joint	8	c
Piston	Screw	2	b
Piston	Gasket	3	b
Conn.rod	Screw	4	b
Conn.rod	Bolt	4	b
joint	Screw	10	b
joint	Bolt	20	b

Table 6 **- Table "PRISE"**

Sparkplug	10
screw	2
gasket	3
bolt	2

Next, we would like to present the Visual Prolog program for solving the component parts problem presented in the book [10] with the purpose of analyzing it for errors and omissions, with subsequent correction and extension.

```
lookfor(Prod,IL,FL,Factor):-
        findall
        ( Pr1,Qty,Flag),
        comp(Prod, Pr1,Qty,Flag), CL),
        inquire(CL,IL,FL,Factor).
inquire([],List,List,_).
inquire([(Pr1,Qty,Flag)|TC],List,FL,Fact):-%%if%%
((Flag==c,  !,
Fact=Fact is Fact*Qty,
lookfor(Pr1,List,ML,Fact1));
                %%elseif%%
(member((Pr1,_),List), !,
update(List,ML,Pr1,Qty,Fact));
                %%else%%
       (Q1 is Qty*Fact,
      add((Pr1,Q1),List,ML),
                %%anyway%%
      inquire(TC,ML,FL,Fact).
update([ (N,Q1)|T],[(N,Q2)|T],N,Q,Fact):-
        Q2  is Q1+Q*Fact.
 update([H|T],[H|T1],N,Q,Fact):-
        update(T,T1,N,Q,Fact).
  add(E,[],[E]).
  add(E,[H,T],[E|[H|T]]).
  member(H, [H|_]).
   member(E,[ _|T]):- member(E,T).

   costs(Prod,Cost):- lookfor(Prod,[],FL,1),
                      examine(FL,Cost).
examine([],0).
 examine([E1,Numb)|T],Cost):-
            price(E1,Cost1),
            examine(T,Cost2),
            Cost = Cost1*Numb+Cost2.
goal
%lookfor(engine,[],FL,1).
costs(engine,Cost).
```

Fig. 2 Prolog program for solving the problem of components
***Source*:** [10]

We give an explanation of the main predicates of the program (Fig. 2). The comp predicate (prod1, prod2, qty, flag) states that the product prod1 contains the qty of instances of the components prod2. The flag flag takes two values: “b” in the case when the prod2 component is defined by a leaf node (simple) “C” in the case when it is complex, that is, the inner top of the tree.

The program for finding elementary components is launched using the lookfor goal with the variable Prod, which determines the name of the product, must be analyzed.

The variable IL, which defines the list when the program starts, takes the value of a blank list, and the Factor variable determines the number of products that need to be exploded. In our case, it has the value 1.

The variable FL is a list and is used to place the results of the program. Each element in this list is complex and defines the name of the component and its quantitative input in the product. For example, in order to recognize in a quantitative measure the components from which the engine is composed, it is enough to start the target:

? lookfor(engine,[],FL,1).

The Lookfor Rule uses the Findall special predicate to find the filial (child) components Pr1 for the parent product or the Prod component, together with the value of the Flags attribute and the number of their occurrences Qty.

The rule inquire performs decomposition of the Pr1 component in the case when Flag is equal to "c", or (otherwise) modifies the number of occurrences in the original list or adds a new element to it with the definition of a quantitative occurrence.

The predicate costs calculates the total cost of the given component.

Note that the book contains a request for the execution of this program and the result of its work. It seems that it is a well-established and working program. But with a more thorough consideration of it, there are doubts about its performance, regardless of the developed qualitative structure, determined at a high level of recursive procedures, their content and implementation.

Here are some errors and problems:

a lot of typos, for example, it is written Fact, but it is necessary Fact1; it is necessary Fl, and there is Ml; need | and worth a coma

more significantly, the built-in predicate Findall requires a task in it of one input variable or structure for the formation of a list, but in the program three variables of the various types are used;

indefinite domains and predicates, but in the version of the Prolog system in which the program was then developed, it might not have been necessary to do it. In today's conditions, in particular, in the Visual Prolog environment, the program will not work and therefore there is a need for its refinement;

the program partially determines the ways of propulsion the algorithm on the tree components nodes, and when moving in the opposite direction from the leaves-nodes to the root node are generally not defined resulting in the wrong answer;

when propulsing the algorithm on the node tree components, the size of the ancestor node in some cases is not calculated at all, which also leads to an incorrect answer.

The authors of the program (Pic. 3) have corrected the following errors.

For the correct organization of the Findall operator, the domains component object strukt3 = struk3 (prod, kol, priz), in which its elements respectively determine the value of the component name, quantity and attribute, are either simple or complex. Based on this object, the list-object list3 and the CL variable were created.

The program also uses the composite object strukt2 = struk2 (prod, kol) and the list2 object list. This object is used in the defining of variables for the formation and placement of output data in the lists of IL, FL and others.

To form a list of nodes for advancing the component tree, the conv predicate (CL, STV, STECV) was created, which performs the concatenation of a part of the list of unprocessed specific top-level nodes STV and a list of certain nodes of the lower level CL formed by a special Findall predicate.

To calculate the component weights (as the product of the number of components of the current node and the components of the ancestor nodes) two gorp predicates (FACTOR, CL, ITOG) and cong (STG, ITOG, STECG) were created, the first of which generates a list of nodes horizontally at the current level and calculates their weights are in the ITOG variable, and the second adds the ITOG list to the main STG list. Fig. 4 shows the request for the formation of the list of components and their quantity required to create the product (engine).

```
/*****************************************************************
        Copyright (c) 1984 - 2000 Prolog Development Center A/S
*****************************************************************/domains
up,down=symbol
dup=up;down
prod = symbol
```

```
    priz=symbol
    kol,ff,cena=integer
    struk3=struk3(prod,kol,priz)
    struk2=struk2(prod,kol)
    list3=struk3*
    list2=struk2*
    predicates
      comp(prod,struk3)
      lookfor(prod,list3,list2,list2,list2,ff)
      inquire(list3,list2,list2,list2,ff,dup)
      member(struk2,list2)
      update(list2,list2,prod,kol,ff)
      add(struk2,list2,list2)
      costs(prod,cena)
      examine(list2,cena)
      price(prod,cena)
      conv(list3,list3,list3)
     cong(list2,list2,list2)
     gorp(ff,list3,list2)
    clauses
    lookfor(Prod,STV,STG,IL,FL,FACTOR):-
              findall( P,  comp(Prod,P), CL),
              gorp(FACTOR,CL,ITOG),
              cong(STG,ITOG,STECG),
              conv(CL,STV,STECV),
              inquire(STECV,STECG,IL,FL,FACTOR,down).
    inquire([],SG,List,List,_,_).
    inquire([struk3(Pr1,Qty,FLAG)|TC],SG,List,FL,Fact,Move):- %%if%%
    FLAG=c,Move=down,!,Fact1=Fact*Qty,lookfor(Pr1,TC,SG,List,FL,Fact1);
    FLAG=c,          Move=up,    !,    comp(Y,struk3(Pr1,Qty,FLAG)),     member(struk2(Y,NFACT),SG),
Fact1=Qty*NFACT,  lookfor(Pr1,TC,SG,List,FL,Fact1);
                    %%elseif%%
    member(struk2(Pr1,_),List), !, update(List,ML,Pr1,Qty,Fact),   inquire(TC,SG,ML,FL,Fact,up);
                    %%else%%
            Q1 = Qty*Fact,   add(struk2(Pr1,Q1),List,ML),
                     %%anyway%%
                inquire(TC,SG,ML,FL,Fact,up).
    update([struk2(N,Q1)|T],[struk2(N,Q2)|T],N,Q,Fact):-
              Q2 = Q1+Q*Fact.
     update([H|T],[H|T1],N,Q,Fact):-
              update(T,T1,N,Q,Fact).
      add(E,[],[E]).
      add(E,[H|T],[E|[H|T]]).
       member(struk2(Pr1,Kol),[struk2(Pr1,Kol)|Tail]).
       member(struk2(E,Kol),[ _|Tail]):-
       member(struk2(E,Kol),Tail).
      conv([],L,L).
      conv([X|L1], L2,[X|L3]):-
           conv(L1,L2,L3).
      cong([],S,S).
      cong([X|S1], S2,[X|S3]):-
           cong(S1,S2,S3).
     gorp(Fact,[],[]).
     gorp(Fact,[struk3(Pr1,Qty,Flag)|TC], [struk2(Pr1,Q1)|T]):-
                              Q1=Qty*Fact,
```

```
                    gorp(Fact,TC,T).
costs(Prod,Cost):- lookfor(Prod,[],[struk2(engine,1)],[],FL,1),
                    examine(FL,Cost) , write (FL),nl,
                    write(Cost).
examine([],0).
examine([struk2(E1,Numb)|T],Cost):-    price(E1,Cost1),
          examine(T,Cost2),
          Cost = Cost1*Numb+Cost2.
comp(engine,struk3(sparkplug,4,b)) .
comp(engine,struk3(cylinder,4,c)) .
comp(engine,struk3(crankshaft,1,c)).
comp(cylinder,struk3(piston,1,c)).
comp(cylinder,struk3(conn_rod,1,c)).
comp(piston,struk3(screw,2,b)).
comp(piston,struk3(gasket,3,b)).
comp(conn_rod,struk3(screw,4,b)).
comp(conn_rod,struk3(bolt,4,b)).
comp(crankshaft,struk3(joint,8,c)).
comp(joint,struk3(screw,10,b)).
comp(joint,struk3(bolt,20,b)).
 price(sparkplug,10).
 price(screw,2).
 price(gasket,3).
 price(bolt,2).
goal
%lookfor(engine,[],[struk2(engine,1)],[],FL,1).
costs(engine,Cost).
```

Fig. 3 The final working program of the solution of the problem of component products

```
[struk2("bolt",176),struk2("gasket",12),struk2("screw",104),struk2("sparkplug",4)]
636Cost=636
1 Solution
```

Fig. 4 Result of implementation of the program for request costs (engine, Cost)

Conclusion. Considered two approaches to the solution of the problems of components indicate their relevance and the need to use in solving the problems of this class.

In the first case, the information base requires more advance preparation. The algorithm of the solution and the program built on its basis is based on a larger number of arrays, increases the memory expenses for their preservation and complicates the development and debugging of the program. But at the same time, from the point of view of the ease of perception of the work of the program, it increases its value.

In the second case, the high power of the Visual Prolog language allows you to form concise programs of high expressiveness.

The power of Visual Prolog is particularly evident in the tasks of processing hierarchical and multi-layered structures. A good example of this is the task in this article.

References

1. Adamenko, A. N., Kuchkov, A. M. (2003). *Logicheskoe prohrammirovanye i Visual Prolog [Logic programming and Visual Prolog].* Saint Petersburg: BHV - Petersburg [in Russian].

2. Biletsky, O. B., Mikhailov, V. S. (1983). *Orhanyzatsyonno-tekhnolohycheskye osnovy ASU v stroytelstve* [*Organizational and technological foundations of the ICS in construction]*. Kyiv: Budivelnik [in Russian].

3. Bondarev, V. N., Ada, F. G. (2002). *Artificial Intelligence [Iskusstvennyi intellekt].* Sevastopol: Publishing House of

Література

1. Адаменко, А. Н. Логическое программирование и Visual Prolog / А. Н. Адаменко, А. М. Кучков – СПб. : БХВ – Петербург, 2003. – 992 с.

2. Билецкий, О. Б. Организационно-технологические основы АСУ в строительстве / О. Б. Билецкий, В. С. Михайлов. – К. : Будівельник, 1983. – 120 с.

3. Бондарев, В. Н. Искусственный интеллект: учебное пособие для ВУЗов / В. Н. Бондарев, Ф. Г. Аде. – Севастополь : Изд-во СевНТУ, 2002. – 615с.

4. Bratko, I. Prolog. Programming for Artificial Intelligence.

ІННОВАЦІЇ

SevNTU [in Russian].
4. Bratko, I. (2004). *Prolog. Programming for Artificial Intelligence*. London, United Kingdom: «Addison Wesley» [in English].
5. Glybovets M. M., Oletsky, O. V. (2002). *Shtuchnyi intelekt [Artificial intelligence].* Kyiv: Publishing House "KM Academia" [in Ukrainian].
6. Kavun, S. V., Korotchenko, V. M. (2007). *Systemy shtuchnoho intelektu [Artificial Intelligence Systems].* Kharkiv: KhNEU [in Ukrainian].
7. Lytvyn, V. V., Pasichnyk, V. V., Yatsishyn, Y. V. (2009). *Intelektualni systemy [Intelligent Systems]*. Lviv: Novyi Svit [in Ukrainian].
8. Russel, S. G., Norvig, P. (2006). *Artificial Intelligence. A modern approach*. New Jersey: «Upper Saddle River» [in English].
9. Luger, G. F. (2008). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. London: «Addison Wesley» [in English].
10. Ceri, S., Gottlob, G., Tanca, L. (1990). *Logic Programming and Databases (Surveys in Computer Science).* (Softcover reprint of the original 1st ed.). Springer-Verlag Berlin Heidelberg [in English].

London, United Kingdom: «Addison Wesley», 637 (2004).
5. Глибовець, М. М. Штучний інтелект: підруч. для студ. вищ. навч. закладів / М. М. Глибовець, О. В. Олецький. – К. : Вид. дім «КМ Академія», 2002. - 366 с.
6. Кавун, С. В. Системи штучного інтелекту: навч. посіб./ С. В. Кавун, В. М. Коротченко. – Харьків : ХНЕУ, 2007. - 320с.
7. Литвин, В. В. Інтелектуальні системи: підручник / В. В. Литвин, В. В. Пасічник, Ю. В. Яцишин. – Львів : Новий світ, 2009. – 406 с.
8. Russel, S. G., & Norvig, P. *Artificial Intelligence. A modern approach*. New Jersey, USA: «Upper Saddle River», 1408 (2006).
9. Luger, G.F. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. London, United Kingdom: «Addison Wesley», 863 (2008).
10. Ceri, S., Gottlob, G., Tanca, L. *Logic Programming and Databases (Surveys in Computer Science)* Softcover reprint of the original 1st ed, Springer-Verlag Berlin Heidelberg (1990).

Received for publication 17.06.2019

Бібліографічний опис для цитування :
Chubaievskyi, V. I. Technologies of multilevel structures modelling on the example of the problem of completing products / V. I. Chubaievskyi, K. O. Palahuta, A. M. Desiatko // Науковий вісник Полісся. – 2019. - № 2 (18). – С. 6-14.

Чубаєвський Віталій Іванович — кандидат політичних наук, доцент кафедри програмної інженерії та кібербезпеки, заступник начальника Департаменту інформаційно-аналітичної підтримки – начальник управління розвитку інформаційних технологій Національної поліції України, полковник поліції, Київський національний торговельно-економічний університет;
https://orcid.org/0000-0001-8078-2652;
E-mail:chubaievskyi_vi@knute.edu.ua;

Chubaievskyi Vitaliy Ivanovich — Candidate of Political Sciences, Associate Professor at the Department of Software Engineering and Cybersecurity, Deputy Head of the Department of Information and Analytical Support - Head of the Department of Information Technology Development of the National Police of Ukraine, Police Colonel, Kyiv National University of Trade and Economics;
https://orcid.org/0000-0001-8078-2652;
E-mail:chubaievskyi_vi@knute.edu.ua;

Палагута Катерина Олексіївна — кандидат економічних наук, доцент, доцент кафедри програмної інженерії та кібербезпеки, Київський національний торговельно-економічний університет;
https://orcid.org/0000-0003-1167-9509;
ResearcherID:N-2928-2016;
E-mail:palagutaea@ukr.net;

Palahuta Katerina Alekseevna — Candidate of Economic Sciences, Associate Professor at the Department of Software Engineering and Cybersecurity, Kyiv National University of Trade and Economics;
https://orcid.org/0000-0003-1167-9509;
ResearcherID:N-2928-2016;
E-mail:palagutaea@ukr.net

Десятко Альона Миколаївна — старший викладач кафедри програмної інженерії та кібербезпеки, Київський національний торговельно-економічний університет;
https://orcid.org/0000-0002-2284-3418;
ResearcherID:N-2873-2016;
E-mail:desyatko@knute.edu.ua;

Desiako Alona Mykolayivna — Senior Lecturer at the Department of Software Engineering and Cybersecurity, Kyiv National University of Trade and Economics;
https://orcid.org/0000-0002-2284-3418;
ResearcherID:N-2873-2016;
E-mail:desyatko@knute.edu.ua.