UDC 004.415.2

METHOD FOR AUTOMATED CLASS TRANSFORMATION UNDER CONDITIONS OF INCOMPLETE ATTRIBUTE DEFINITION

Kungurtsev O. B. – PhD, Professor of the Software Engineering Department, Odessa Polytechnic National University, Odessa, Ukraine. ROR: https://ror.org/05xaz0w84. ORCID: http://orcid.org/0000-0002-3207-7315.

Antonov A. O. – Student of the Software Engineering Department, Odessa Polytechnic National University, Odessa, Ukraine. ROR: https://ror.org/05xaz0w84. ORCID: https://orcid.org/0009-0008-3326-6856.

Novikova N. O. – PhD, Associate Professor of the Department of Technical Cybernetics and Information Technologies named after professor R. V. Merct, Odessa National Maritime University, Odessa, Ukraine. ROR: https://ror.org/05qze6v15. ORCID: https://orcid.org /0000-0002-6257-9703.

ABSTRACT

Context. Finding solutions to problems with an incomplete set of necessary data is a relevant problem in various applied research. In object-oriented programming, a similar problem arises when it is necessary to create a class object in the absence of some data regarding its initialization.

Objective. If there is enough data to solve a certain set of problems, it is advisable to create an object of the corresponding class. Problems arise when not all attributes of the created object have values. This can lead to errors in the operation of the object when calling methods that use undefined attributes. The aim of the work is to develop a mechanism that provides blocking of methods that are not ready to work, as well as their gradual unlocking as values for the attributes are defined.

Method. A mathematical model of the class is proposed, which allows distinguishing two subsets of attributes that do not have values at the time of creating the class object. A method has been developed that automatically converts the source class, adding a mechanism for locking/unlocking certain methods depending on the presence or absence of attribute values that these methods directly or indirectly use.

Result. The Class Transformer software product was created, which allowed a series of experiments to be conducted that confirmed the effectiveness of the research. The experiments showed a significant reduction in class conversion time compared to performing this work in the traditional way.

Conclusions. The proposed method for automated class conversion in conditions of incomplete attribute definition, which, on the one hand, allows you to start working with the object without waiting for the moment of complete data definition, and on the other hand, reduces the time for class conversion. The method can be used for various object-oriented programming languages.

KEYWORDS: object-oriented programming, classes, undefined attributes, method locking, parsing, class conversion.

ABBREVIATIONS

OOP – object-oriented programming.

NOMENCLATURE

attrName is an attribute identifier;
attrType is an attribute type;
cHead is a class heading;
cName is a class name;
fName is a method name;
sArgs is a set of method arguments;
sAttr is a set of class attributes;
sConstr is a set of class constructors;
sMeth is a set of class methods;
sOperand is a set of operands;
reType is a type of the return value;
TClassLoad is a time to load the class into the ClassTransformer;
TFConstr is a time to form a constructor;
TFMethLock is a time to form methods with blocking;
TFMtxFA is a time for formation of the MtxFA matrix;
TFMtxFF is a time for forming the MtxFF matrix;
TFSet is a time to form set() methods;
TSAttrInMeth is a time to search for attributes in methods;
TSMethInMeth is a time searching in methods for other methods that use undefined attributes;
TUndefAttrRendering is a time to form a set of defined attributes;
var is a variable;
TProgrammRunning is a time to transform the class into a *ClassTransformer*.

INTRODUCTION

Object-oriented software development technologies are currently widely employed [1, 2]. Object-oriented software products find application across diverse fields of activity [3, 4]. Regardless of the specific domain for which a software product is created, users frequently face the necessity of making decisions under conditions of incomplete or imprecise input data. It is desirable that the software to some extent helps in solving such problems. This paper examines the problem of organizing the operation of object-oriented programs when dealing with incomplete input data. The fundamental elements of such a program are the software class and the object instantiated from it. The developer of a software class must guarantee the full functionality of the class object upon its creation. This implies the necessity of possessing the complete set of data (attribute values) at the moment of class object initialization. This requirement may conflict with the actual technologies within the domain. For instance, an incomplete dataset might suffice to execute a certain portion of the required functions, potentially for an extended pe-

DOI 10.15588/1607-3274-2026-2-15

riod. Awaiting the availability of complete data before object creation leads to losses of various natures within the application domain of the software product. A potential resolution might involve creating a multitude of highly specialized classes; however, this results in the complication of the software structure and reduces the potential for code reuse. Thus, a problem exists regarding the utilization of objects instantiated from software classes under conditions of incomplete initialization data.

The object of study is the processes of determining the functionality of software systems under conditions of incomplete data.

The subject of study is models of software classes and the mechanisms of operation of their objects under conditions of incomplete data on the meaning of attributes in OOP.

The research aims to reduce the time it takes to process events in software systems by creating class objects under conditions where not all attribute values are known. To achieve this goal, the following tasks must be solved:

– to develop a software class model that takes into account uncertain values for some attributes;

– to develop a method for transforming a software class into a form where its objects can operate under conditions of incomplete attribute value data;

– to evaluate the effectiveness of the proposed solutions.

1 PROBLEM STATEMENT

Let us represent the software class as a set of methods and attributes:

$$c=<sAttr, sMeth>.$$

Any method is a function of some subset of the class attributes:

$$Meth_i = f(sAttrM_i), \text{ where } sAttrM_i \in sAttr.$$

Suppose that at a specific time instant t_0, only a portion of the attributes can receive values:

$$sAttr = sAttrWithVal \cup sAttrWithoutVal.$$

Then, the attribute subset required for a method $Meth_i$ can be partitioned as:

$$sAttrM_i = sAttrWithValM_i \cup sAttrWithoutValM_i.$$

If a class object *cObj* is created at time t_0 any method $Meth_i$ must be blocked $Meth_i \Rightarrow BlockMeth_i$ under the condition that $sAttrWithoutValM_i \neq 0$.

If, during the operation of the class object *cObj*, an attribute $attr_j$ receives a value (i.e., $attr_j \in sAttrWithValM_i$) and the corresponding subset of missing attributes for a method becomes empty (i.e., $sAttrWithoutValM_i = 0$), then the corresponding method must become unblocked ($BlockMeth_i \Rightarrow Meth_i$).

2 REVIEW OF THE LITERATURE

The formulated problem can be categorized as a self-adaptive software creation task. In this context, the paper [5] is of interest, as it provides an analysis of adaptive system organization principles and highlights the significance of feedback. However, this study does not propose solutions at the level of concrete programming technologies.

The study [6] addresses the issue of incomplete data, suggesting that missing data should be imputed based on the history of their values to maintain the statistical significance of the entire dataset. When applied to classes, this approach could lead to significant errors during the execution of individual class methods. A somewhat different situation is analyzed in the paper [7], where data incompleteness is attributed to the presence of noise and outliers. The recommendations pertain to isolating useful data from that distorted by interference. Such a solution might find application in object-oriented systems, but not at the level of object manipulation, but rather at the stage of input data preprocessing.

If missing data are considered as data with an undefined type, then the study [8] is relevant, as it demonstrates how to maintain program quality during the gradual definition of types. However, existing programming languages with dynamic typing [9] do not allow transitioning from some initially established type to a static (finally determined) one.

Since the initial data for class object creation are its attribute values, research [10], which proposes defining the effectiveness of class attributes in terms of executing its primary functions, is of interest. If the proposed idea is conceptualized as defining the minimal set of attributes required for class object instantiation, it could be applicable to this research.

The resolution of the incomplete data problem for object initialization is intrinsically linked to the delineation of class elements such as attributes and methods. The article [11] examines the syntactic analysis of a class, but the primary focus is on determining the performance of a novel deterministic algorithm that executes a full code analysis, which is excessive for the objectives of the current study. Parsing is also addressed in the paper [12]; however, its authors only reveal the object-oriented structure of the program without detailing the class elements. A partial analysis of class code is implemented in the study [13], which can be utilized in this investigation.

Solving the issue of indeterminate attributes necessitates the formalization of class description, i.e., the creation of its model. For the last two decades, the UML language has remained relevant for constructing models and diagrams in object-oriented technologies [14]. Nevertheless, UML class models are primarily intended to represent class interactions, rather than the interactions of their constituent elements. An attempt to enhance UML was proposed in the study [15]. The author points to the shortcomings of UML and suggests using the new modeling language Matching Machine, which aims to partially link attributes and the actions performed upon them. However, this language cannot currently be used to discover and model the complete interaction between attributes and methods within a software class. The study [16] proposes a mathematical model of a software class used for finding and comparing classes that can be related by inheritance.

DOI 10.15588/1607-3274-2026-2-15

This model allows for the identification of the internal elements of a class. Despite the model's narrow, specialized use, the principles of its construction can be employed in this research. The mathematical model of a class proposed in the paper [13] also deserves attention, where the primary focus is on working with attributes, but unlike this study, only within the context of class composition.

3 MATERIALS AND METHODS

Class model.

The class is represented as a tuple:

$$c = < cHead, sAttr, sMeth >. \quad (1)$$

The class header is represented as a tuple:

$$cHead = < cName, cName_1 >,$$

where $cName$ – name of the source class; $cName_1$ – name of the transformed class.

Each attribute from the set $sAttr$ is represented in the form:

$$Attr = < attrName, attrType >.$$

The set of methods $sMeth$ is represented by the tuple:

$$sMeth = < sFunc, sConstr, dest >. \quad (2)$$

Any element from $sMeth$ has the form:

$$Meth_i = < fName_i, mArgs_i, retType_i, sOperator_i >, \quad (3)$$

where $retType_i$ – the type of value that the method returns.

Any operator is represented as a set of operands (variables, constants, function calls):

$$Operator = sOperand.$$

Each operand can be of one of three types:

$$operand_i = const \mid var \mid methCall,$$

where $const$ – constant; var – variable; $methCall$ – class method call.

From the perspective of method control, operands of two types are of interest: variables, which are class attributes, and function calls. In this case, formula (3) takes the form:

$$Meth_i = < fName_i, mArgs_i, retType_i, sAttrM_i, sMethCallM_i >, \quad (4)$$

where $sAttrM_i$ – attributes used in the method $Meth_i$; $sMethCallM_i$ – class methods used in the method $Meth_i$.

All attributes can be divided into two categories from the viewpoint of their mutability during the class object's operation:

$$sAttr = sImmutableAttr \cup sMutableAttr. \quad (5)$$

The subset $sImmutableAttr$ includes constant attributes, i.e., attributes whose value cannot change after object initialization (e.g., an employee's date of birth). The subset $sMutableAttr$ includes attributes whose value can change after object initialization.

From another perspective, all attributes can be divided into two groups depending on the definiteness of their values during the class object's operation:

$$sAttr = sAttrWithVal \cup sAttrWithoutVal. \quad (6)$$

The subset $sAttrWithVal$ includes attributes that have values at the moment of class object creation. The subset $sAttrWithoutVal$ includes attributes that may remain uninitialized for some time after class object creation. Thus:

$$sAttrWithoutVal = s1ImmutableAttr \cup s1MutableAttr, \quad (7)$$

where $s1ImmutableAttr \in sImmutableAttr$; $s1MutableAttr \in sMutableAttr$.

The introduction of the concept of indeterminate attributes necessitates the definition of indeterminate methods ($sUndefMeth$), which utilize these attributes.

If $\exists\ attr_j \mid attr_j \in sAttrWithoutVal \wedge attr_j \in sAttrM_j$, then the class method $Meth_i \in sUndefMeth$.

Method for class transformation via the creation of a method locking / unlocking mechanism.

Input data: A given class from which it is meaningful to instantiate objects despite incomplete information regarding attribute values. We assume that the class undergoing transformation is not inherited from another class.

First Stage. Formation of defined and indeterminate attribute sets. We analyze the set $sAttr$ and form the subset of attributes $sAttrWithoutVal$ that lack values at the moment of class object creation. From $sAttrWithoutVal$, we distinguish the subset of attributes $sImmutableAttr$, which remain unchanged during object operation, and the subset of mutable attributes $s1MutableAttr$.

Second Stage. Determination of indeterminate methods (those using indeterminate attributes). For every method $Meth_i \in sMeth$ we determine the subset of attributes it utilizes as operands:

$$sAttrM_i \in sAttr.$$

We isolate the subset of value-less attributes from $sAttrM_i$:

$$sAttrWithoutValM_i \in sAttrM_i \wedge sAttrWithoutValM_i \in sAttrWithoutVal.$$

DOI 10.15588/1607-3274-2026-2-15

Every method $Meth_i$ for which $sAttrWithoutValM_i \neq 0$ is included in the set of indeterminate methods *sUndefMeth*.

Third Stage. Identification of methods utilizing indeterminate methods. A method that invokes an indeterminate method is itself considered indeterminate. For every method $Meth_i \in sMeth$ we determine the subset of methods it utilizes as operands: $sMethCallM_i \in sMeth$.

We isolate the subset of references to indeterminate methods from $sMethCallM_i$:

$$sUndefMethM_i \in sMethCallM_i \wedge sUndefMethM_i \in sUndefMeth.$$

Every method $Meth_i$ for which $sUndefMethM_i \neq 0$ is then included in the set of indeterminate methods *sUndefMeth*.

We define an element of the *sUndefMeth* as:

$$sUndefMeth_j = < fName_j, sAttrWithoutValM_j, sUndefMethM_j >.$$

Fourth Stage. Creation of method control matrices, method modernization. The mechanism for controlling method lockings, dependent on attribute states, is conveniently represented using a matrix *MtxFA* (Table 1).

Table 1 – *MtxFA* matrix for method locking control based on attributes

Methods	Attributes					
	a_1	a_2	a_3	…	a_{k-1}	a_k
f_1	*true*	*false*	*true*	…	*true*	0
f_2	*false*	*true*	*true*	…	*true*	*true*
…	…	…	…	…	…	…
f_q	*true*	*true*	*false*	…	*false*	*true*

The matrix presents the states of attributes a_1, a_2, a_3,..., a_{k-1}, a_k from the set *sAttrWithoutVal* and the names of functions f_1, f_2,..., f_q from the set *sUndefMeth*. Each cell of the matrix can have the value *false* (blocking) or *true* (no blocking).If, at the moment of class object creation, the entry $MtxFA_{i,j}$ contains the value *false*, it signifies that the attribute a_j lacks a value and consequently blocks the function f_i. Conversely, if the cell $MtxFA_{i,j}$ contains the value *true*, it indicates that the attribute a_j is not blocking for the function f_i.

As the class object operates, the attributes from the set *sAttrWithoutVal* progressively acquire values, and the locks are removed from the methods belonging to the set *sUndefMeth.*

The mechanism for controlling method lockings dependent on the state of other methods is conveniently represented using the matrix *MtxFF* (Table 2).

Table 2 – *MtxFF* matrix for method locking control based on methods

Methods	Methods			
	f_1	f_2	…	f_t
f_1	*true*	*true*	…	*false*
f_2	*false*	*true*	…	*true*
…	…	…	…	…
f_q	*true*	*false*	…	*true*

In the matrix *MtxFF* , the methods (functions) belonging to the set *sUndefMeth* are listed both vertically and horizontally. If the entry $MtxFF_{i,j}$ contains the value *false*, it implies that the method f_j is indeterminate, and consequently, the method f_i is blocked.

As the class object operates, the locks are removed from the methods within the set *sUndefMeth.*

The result of executing this stage is the creation of the locking control matrices and the modernization of the set of indeterminate methods:

$$sUndefMeth \Rightarrow sModUndefMeth.$$

Fifth Stage. Adjustment of class constructor code. Elements associated with indeterminate attributes are removed from every constructor.

We represent the set of constructors as:

$$sConstr = \{constr_i\}; \quad i = 1, |\, sConstr\,|.$$

A constructor is represented as a set of arguments and operators:

$$constr_i = \{< sArgs, sOperator >\}.$$

We define the set of operators that do not utilize the indeterminate attributes *sAttrWithoutVal*:

$$sDefOperator = \{\, operator_j \,|\, sAttrWithoutVal_k \notin_{op} operator_j \,\},$$
$$j = 1, |\, sOperator\,|;\ \ k = 1, |\, sAttrWithoutVal\,|;$$

where the relation $\notin_{op}$ denotes the non-use of an attribute in the body of the operator.

sDefOperator represents the constructor's new set of operators.

We then define the constructor arguments that do not utilize attributes from *sAttrWithoutVal:*

$$sDefArg = \{\, args_j \mid args_j \in_{op} sDefOperator \,\};$$
$$j = 1, |\, sArgs\,|.$$

sDefArg represents the constructor's new set of arguments. The operations considered are repeated for all constructors. The result of executing this stage is the modernized set of constructors: $sConstr \Rightarrow sModConstr$.

Sixth Stage. Adjustment of attribute value setting methods. For every attribute $attr_i \in sAttrWithoutVal$ a *set*() method must be created to assign its value during the object's operation. Furthermore, if $attr_i \in s1MutableAttr$, the $set()_i$ method can be used after the initialization of $attr_i$. If, however, $attr_i \in s1ImmutableAttr$, the $set()_i$ method must be blocked following attribute initialization.

DOI 10.15588/1607-3274-2026-2-15

We denote the relationship between an attribute and its value-defining method as:

$$\forall\ attr_i \in s1MutableAttr \rightarrow set()_i;$$
$$\forall\ attr_i \in s1MutableAttr \rightarrow setB()_i.$$

Here, the symbol $\rightarrow$ denotes the requirement to create the method. $set()_i$ is a standard attribute value setting method. $setB()_i$ is a method for setting the value of attribute $attr_i$, which is automatically blocked after its invocation and is not used subsequently.

The result of executing this stage is the modernized set of *set*() methods:

$$sSetMeth \Rightarrow sModSetMeth,$$

where *sSetMeth* is the set of *set*() methods prior to modernization.

Seventh Stage. Formation of the modernized class *cName*1. In the modernized class, the description of attributes and methods that do not depend on the indeterminate attributes *sAttrWithoutVal* remains unchanged. The method control matrices *MtxFA* and *MtxFF*, the set of modernized constructors *sModConstr,* the set of modernized indeterminate methods *sModUndefMeth*, and the set of value-setting methods *sModSetMeth* are added.

Case where class *cName* is derived from class *cParent*:

Variant 1. Code for *cParent* is available

First Stage. The methodology proposed above is applied to the class *cParent.*

Second Stage. The proposed methodology is then applied to the derived class *cName.*

Variant 2. Code for class *cParent* is inaccessible.

First Stage. A set of indeterminate attributes *sAttrWithoutValP* for class *cParent* is constructed. This set is unioned with the set of indeterminate attributes of the derived class.

Second Stage. A set of indeterminate methods *sUndefMethP* for class *cParent* is constructed (based on analysis or possibly experimentation). Every such method in class *cParent* is overridden by a wrapper method in the derived class.

Third Stage. In the invocation of the *cParent* class constructor, fictitious, yet permissible, values are assigned for the indeterminate attributes. These values can be arbitrary since they will subsequently be blocked.

Fourth Stage. For all attributes in the set *sAttrWithoutValP, set*() methods are created.

Fifth Stage. The matrices *MtxFA* and *MtxFF* are augmented to include the indeterminate attributes and methods originating from the parent class *cParent*.

4 EXPERIMENTS

To implement the proposed methodology, the software product *Class Transformer* was developed. Figure 1 presents the operational diagram of the *Class Transformer* software product. *Class Transformer* accepts the code of a software class as input. As a result of the code analysis, a list of attributes is extracted and presented to the programmer for the identification of the indeterminate attributes *sAttrWithoutVal.* Based on the identified attributes and the analysis of the class method code, the set of indeterminate methods *sUndefMeth* is formed. The matrices *MtxFA* and *MtxFF* are then created to control the methods within the set *sUndefMeth* based on the values of attributes and other indeterminate methods.

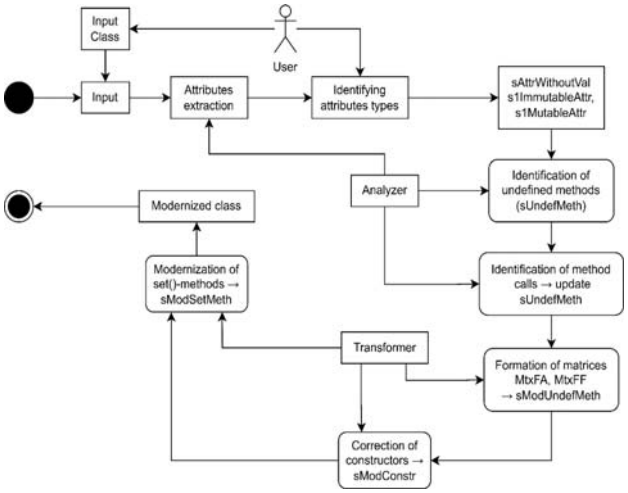

Figure 1 – Operational diagram of the *ClassTransformer* software product

Operators are introduced into all methods *sUndefMeth* to enable the blocking of their execution if, at the time of invocation, not all utilized attributes or methods are defined. The constructor code is adjusted based on *sAttrWithoutVal. set*() methods are added to the class code if they were absent in the source code. After performing all the listed operations, the output of the *Class Transformer* yields a transformed class that is capable of operating under conditions of attribute incompleteness and guarantees the full functionality of the class object.

Figure 2 displays the graphical user interface window for forming the list of indeterminate attributes.

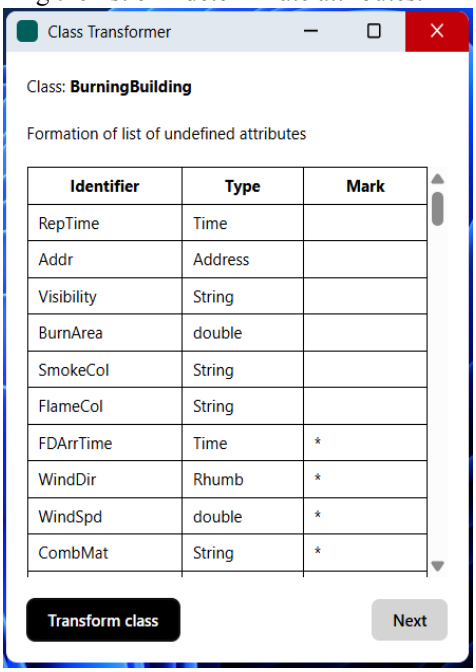

Figure 2 – Formation of the list indeterminate attributes

DOI 10.15588/1607-3274-2026-2-15

5 RESULTS

The validation of the proposed method and software consists of determining the operability of the software product and comparing the time required for class transformation in both manual and automated modes.

To conduct the study, six classes were created, containing 5, 10, and 15 indeterminate attributes respectively, for the subject domains of "Tirefighting" and "Healthcare". The number of methods was set at 3–4 per attribute. The list of indeterminate attributes was established a priori.

To determine the time required for manual class transformation, eight students who had successfully completed the course "Object-Oriented Programming" were engaged.

The time for manual transformation can be represented by the following formula:

$$TManualTrasfer = TSAttrInMeth + TSMethInMeth + TFMtxFA + TFMtxFF + TFMethLock + TFConstr + TFSet, \quad (8)$$

where *TSAttrInMeth* – time spent on searching for attributes within methods; *TSMethInMeth* – time spent on searching within methods for other methods that utilize indeterminate attributes; *TFMtxFA* – time spent on forming the *MtxFA* matrix; *TFMtxFF* – time spent on forming the *MtxFF* matrix; *TFMethLock* – time spent on forming methods with locking mechanisms; *TFConstr* – time spent on forming the constructor; *TFSet* – time spent on forming the *set*() methods.

Figure 3 presents the experimental data for determining the average time required for manual class transformation as a function of the number of indeterminate attributes in the input class. During the experiment, three errors were recorded (one during work with a class containing 10 indeterminate attributes, and two during work with a class containing 15 indeterminate attributes).

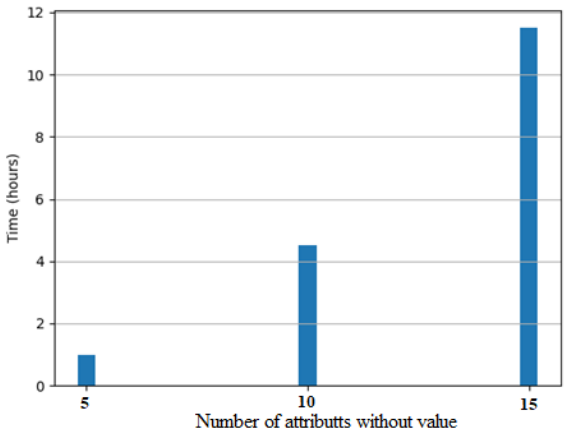

Figure 3 – Time required for manual class transformation

The time for automated transformation can be represented by the following formula:

$$TAutomaticTrasfer = TClassLoad + TUndefAttrRendering + TProgramRunning, \quad (9)$$

where *TClassLoad* – time required to load the class into the *Class Transformer*; *TUndefAttrRendering* – time required to form the set of defined attributes (rendering the list to the programmer for selection); *TProgramRunning* – time required for the class transformation within the *Class Transformer*.

Figure 4 presents the experimental data for determining the time required for class transformation in automated mode as a function of the number of indeterminate attributes in the input class.

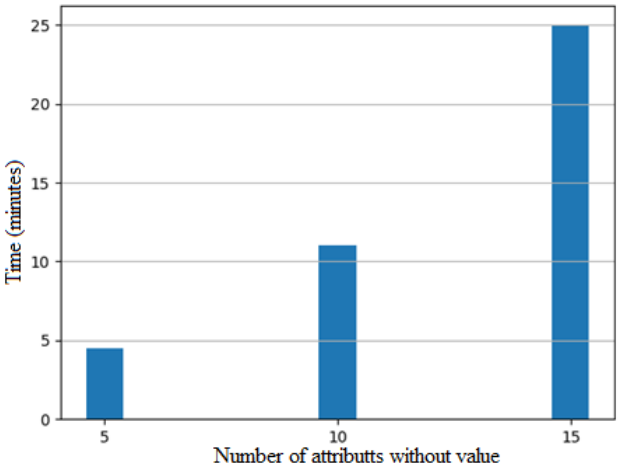

Figure 4 – Time required for class transformation in automated mode

Thus, the validation demonstrated that the application of the automated method reduces the class transformation time by an average of 25 times and effectively prevents errors. The method locking mechanism slows down the operation of the class object. To estimate the time losses, the average execution time of the methods was determined under the condition that all attributes received values. Figure 5 shows the average measurement results for a class object with method locks and a similar object without locks.

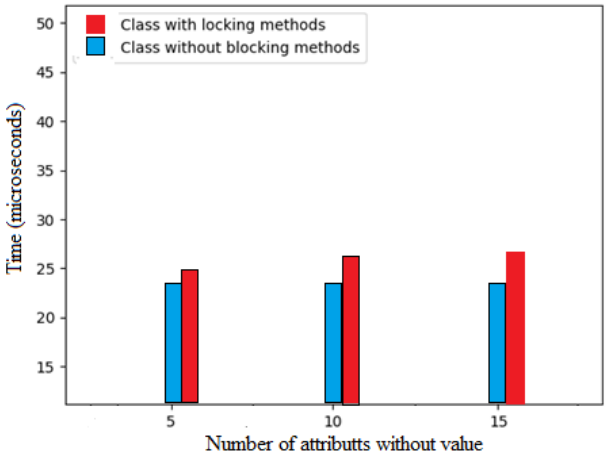

Figure 5 – Average execution time of methods with and without blocking

DOI 10.15588/1607-3274-2026-2-15

6 DISCUSSION

The conducted experiments demonstrated that the application of the proposed method significantly reduces the transformation time of a class designated for working with partially indeterminate (or undefined) attributes. It should be noted that objects of the transformed class operate somewhat slower than objects of a standard class with analogous functionality due to the utilization of the method locking mechanism (Fig. 5). If the problem of replacing one operational class object with another is resolved as values are assigned to the indeterminate attributes, a solution ensuring enhanced performance might be found, particularly when the number of indeterminate attributes is small. Additionally, a challenge exists regarding the transformation of a descendant class (subclass) if the indeterminate attributes belong to the parent class (superclass), for which complete information (code) is unavailable.

CONCLUSIONS

It has been proven that a problem exists in organizing the operation of a class object under the condition of incomplete attribute definition (or indeterminate attributes), the solution to which can improve the operational characteristics of software classes in various object-oriented products.

A software class model is proposed, which allows for the identification of indeterminate attributes, the methods dependent on them, and the methods dependent on other indeterminate methods. The model makes it possible to determine the operational part of the class at each stage of the gradual definition of attribute values.

A method for automated software class transformation has been developed. It involves modifying the class code to enable class objects to adapt to operating conditions with indeterminate attributes.

A software product, *Class Transformer*, has been created to implement the developed method. Experiments were conducted that confirm the effectiveness of the obtained scientific results. It has been established that classes transformed using Class Transformer fully execute their functions, and the automated transformation time is 25 times shorter compared to the “manual” transformation mode.

ACKNOWLEDGEMENTS

We would like to thank our colleagues Anastasia Troynina and Svetlana Zinovatnaya for their active participation in discussions on the use of data in refining its meanings in software systems.

DECLARATIONS

Conflict of interest: The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship, or otherwise, that could affect the research and its results presented in this paper.

Authors contributions: Oleksii Kungurtsev: development of a method for class transformation via the creation of a method locking / unlocking mechanism; Nataliia Novikova: development a class model, development of actions in case where class *cName* is derived from class *cParent*; Artem Antonov: creation of software for experimental confirmation of the effectiveness of the method for class transformation.

Data availability: All data is provided in the article.

Software availability: The manuscript has related software at the link:

https://drive.google.com/file/d/1uhC4ZFzNhH3wc3derRuYaJmmz_t4KT88/view?usp=sharing.

Use of artificial intelligence tools: The authors confirm that they did not use artificial intelligence technologies in creating the submitted work.

REFERENCES

1. Liu W. The Implications of Object-Oriented Analysis and Design. *Global Journal of Engineering, Design & Technology,* 2022, Vol. 11, Iss. 3, P. 157. DOI: 10.35248/2319-7293.22.11.157.
2. Agu S., Elugwu F. Object Oriented Programming Approach. A Panacea for Effective Software Development [Electronic resource]. *African Journal of Advanced Science & Technology Research*, 2022, Vol. 6, № 1. Regime of access: https://publications.afropolitanjournals.com/index.php/ajastr/article/view/215 free (date of the application: 30.09.2022). Header from the screen.
3. Perrelli M., Cosco F., Carbone G., et al. On the Benefits of Using Object-Oriented Programming for the Objective Evaluation of Vehicle Dynamic Performance in Concurrent Simulations. *Machines*, 2021, Vol. 9, Iss. 2, P. 41. DOI: 10.3390/machines9020041.
4. Wen P. Y., Chang S. Y. Design and Implementation of Model-Driven Development for Nursing Information System. *Stud Health Technol Inform*, 2022, Vol. 290, pp. 154–157. DOI: 10.3233/SHTI220051. PMID: 35672990.
5. Rashid T. A., Hassan B. A., Alsadoon A. et al. Awareness requirement and performance management for adaptive systems: a survey. *The Journal of Supercomputing*, 2023, Vol. 79(9), pp. 9692–9714. DOI: 10.1007/s11227-022-05021-1.
6. Johnny V., Philip M., Augustine S. Methods to Handle Incomplete Data. *MAMC Journal of Medical Sciences,* 2020, Vol. 6(3), P. 194. DOI: 10.4103/mamcjms.mamcjms_54_20.
7. Sun Z., Gao M., Jiang A. et al. Incomplete data processing method based on the measurement of missing rate and abnormal degree: Take the loose particle localization data set as an example. *Expert Systems with Applications: An International Journal*, 2023, Vol. 216, P. 119411. DOI: 10.1016/j.eswa.2022.119411.
8. New M. S., Licata D. R., Ahmed A. Gradual type theory. *Journal of Functional Programming*, 2021, Vol. 31, P. 21. DOI: 10.1017/S0956796821000125.
9. Cassola M., Talagorria A., Pardo A., et al. A gradual type system for Elixir. *Journal of Computer Languages*, 2022, Vol. 68, Iss. 4, P. 101077. DOI: 10.1016/j.cola.2021.101077.
10. Rashidi H., Azadi F. On Attributes of Objects in Object-Oriented Software Analysis. *International Journal of Industrial Engineering & Production Research*, 2019, Vol. 30(3), pp. 341–352. DOI: 10.22068/ijiepr.30.3.341.

DOI 10.15588/1607-3274-2026-2-15

11. Slivnik B. Context-sensitive parsing for programming languages. *Journal of Computer Languages,* 2022, Vol. 73, P. 101172. DOI: 10.1016/j.cola.2022.101172.
12. Wojszczyk R., Hapka A., Królikowski T. Performance analysis of extracting object structure from source code. *27th International Conference on Knowledge Based and Intelligent Information and Engineering Systems (KES 2023) : proceedings. Procedia Computer Science*, 2023, Vol. 225, pp. 4065–4073. DOI: 10.1016/j.procs.2023.10.402.
13. Kungurtsev O. B., Bondar V. R., Gratilova K. O. et al. Method Automated Class Conversion for Composition Implementation. *Radio Electronics, Computer Science, Control,* 2024, № 2, pp. 142–149. DOI: 10.15588/1607-3274-2024-2-14.
14. Shah K. P., Grant E. S. Towards Verification of UML Class Models using Formal Specification Methods: A Review [Electronic resource]. *Global Journal of Computer Science and Technology*, 2023, Vol. 23, № H1. Regime of access: https://computerresearch.org/index.php/computer/article/view/102296 free (date of the application: 25.10.2025). Header from the screen.
15. Al-Fedaghi S. Classes in Object-Oriented Modeling (UML): Further Understanding and Abstraction. *International Journal of Computer Science and Network Security*, 2021, Vol. 21, № 5, pp. 139–150. DOI: 10.48550/arXiv.2106.00267.
16. Kungurtsev O. B., Vytnova A. I. Determination of Inheritance Relations and Restructuring of Software Class Models in the Process of Developing Information Systems. *Radio Electronics, Computer Science, Control*, 2022, № 4, pp. 98–107. DOI: 10.15588/1607-3274-2022-4-8.

Received 05.11.2025.
Accepted 03.03.2026.
Published 26.06.2026.

УДК 004.415.2

МЕТОД АВТОМАТИЗОВАНОГО ПЕРЕТВОРЕННЯ КЛАСУ ДЛЯ РОБОТИ В УМОВАХ НЕПОВНОГО ВИЗНАЧЕННЯ АТРИБУТІВ

Кунгурцев О. Б. – канд. техн. наук, професор кафедри інженерії програмного забезпечення Національного університету «Одеська політехніка», Одеса, Україна. ROR: https://ror.org/05xaz0w84. ORCID: http://orcid.org/0000-0002-3207-7315.

Антонов А. О. – студент кафедри інженерії програмного забезпечення Національного університету «Одеська політехніка», Одеса, Україна. ROR: https://ror.org/05xaz0w84. ORCID: https://orcid.org/0009-0008-3326-6856.

Новікова Н. О. – канд. техн. наук, доцент кафедри Технічна кібернетика й інформаційні технології ім. професора Р. В. Меркта Одеського національного морського університету, Одеса, Україна. ROR: https://ror.org/05qze6v15. ORCID: https://orcid.org /0000-0002-6257-9703.

АНОТАЦІЯ

Актуальність. Пошук рішень для задач при неповному наборі необхідних даних – актуальна проблема різних прикладних дослідженнях. У об'єктно-орієнтованому програмуванні подібна проблема виникає за необхідності створити об'єкт класу за умови відсутності деяких даних щодо його ініціалізації.

Мета роботи. Якщо даних достатньо для вирішення деякої підмножини завдань, бажано створити об'єкт відповідного класу. Проблеми виникають, коли у створеного об'єкта не всі атрибути мають значення. Це може призвести до помилок у роботі об'єкта під час виклику методів, які використовують невизначені атрибути. Метою роботи є розробка механізму, який забезпечує блокування методів неготових працювати, а також поступове їх розблокування в міру визначення значень для атрибутів.

Метод. Запропоновано математичну модель класу, яка дозволила виділити два підмножини атрибутів, які не мають значень у момент створення об'єкта класу. Розроблено метод, який автоматично перетворює вихідний клас, додаючи механізм блокування/деблокування певних методів залежно від наявності або відсутності значень атрибутів, які ці методи безпосередньо чи опосередковано використовують.

Результати. Створено програмний продукт *ClassTransformer*, який дозволив провести серію експериментів, що підтвердили ефективність досліджень. Експерименти показали істотне скорочення часу перетворення класів порівняно з виконанням цієї роботи традиційним способом.

Висновки. Запропонований метод автоматизованого перетворення класу в умовах неповного визначення атрибутів, який з одного боку дозволяє не чекаючи моменту повного визначення даних починати працювати з об'єктом, а з іншого боку скорочує час на перетворення класу. Метод може бути використаний для різних об'єктно-орієнтованих мов програмування.

КЛЮЧОВІ СЛОВА: об'єктно-орієнтоване програмування, класи, невизначені атрибути, блокування методів, парсинг, перетворення класу.

ЛІТЕРАТУРА

1. Liu W. The Implications of Object-Oriented Analysis and Design / W. Liu // Global Journal of Engineering, Design & Technology. – 2022. – Vol. 11, Iss. 3. – P. 157. DOI: 10.35248/2319-7293.22.11.157.
2. Agu S. Object Oriented Programming Approach. A Panacea for Effective Software Development [Electronic resource] / S. Agu, F. Elugwu // African Journal of Advanced Science & Technology Research. – 2022. – Vol. 6, № 1. – Regime of access: https://publications.afropolitanjournals.com/index.php/ajastr/article/view/215 free (date of the application: 30.09.2022). – Header from the screen.
3. On the Benefits of Using Object-Oriented Programming for the Objective Evaluation of Vehicle Dynamic Performance in Concurrent Simulations / [M. Perrelli, F. Cosco, G. Carbone et al.] // Machines. – 2021. – Vol. 9, Iss. 2. – P. 41. DOI: 10.3390/machines9020041.
4. Wen P. Y. Design and Implementation of Model-Driven Development for Nursing Information System / P. Y. Wen,

DOI 10.15588/1607-3274-2026-2-15

S. Y. Chang // Stud Health Technol Inform. – 2022. – Vol. 290. – P. 154–157. DOI: 10.3233/SHTI220051. PMID: 35672990.
5. Awareness requirement and performance management for adaptive systems: a survey / [T. A. Rashid, B. A. Hassan, A. Alsadoon et al.] // The Journal of Supercomputing. – 2023. – Vol. 79(9). – P. 9692–9714. DOI: 10.1007/s11227-022-05021-1.
6. Johnny V. Methods to Handle Incomplete Data / V. Johnny, M. Philip, S. Augustine // MAMC Journal of Medical Sciences. – 2020. – Vol. 6(3). – P. 194. DOI: 10.4103/mamcjms.mamcjms_54_20.
7. Incomplete data processing method based on the measurement of missing rate and abnormal degree: Take the loose particle localization data set as an example / [Z. Sun, M. Gao, A. Jiang et al.] // Expert Systems with Applications: An International Journal. – 2023. – Vol. 216, – P. 119411. DOI: 10.1016/j.eswa.2022.119411.
8. New M. S. Gradual type theory / M. S. New, D. R. Licata, A. Ahmed // Journal of Functional Programming. – 2021. – Vol. 31. – P. 21. DOI: 10.1017/S0956796821000125.
9. A gradual type system for Elixir / [M. Cassola, A. Talagorria, A. Pardo, et al.] // Journal of Computer Languages. – 2022. – Vol. 68, Iss. 4. – P. 101077. DOI: 10.1016/j.cola.2021.101077.
10. Rashidi H. On Attributes of Objects in Object-Oriented Software Analysis / H. Rashidi, F. Azadi // International Journal of Industrial Engineering & Production Research. – 2019. – Vol. 30(3). – P. 341–352. DOI: 10.22068/ijiepr.30.3.341.
11. Slivnik B. Context-sensitive parsing for programming languages / B. Slivnik // Journal of Computer Languages. – 2022. – Vol. 73. – P. 101172. DOI: 10.1016/j.cola.2022.101172.
12. Wojszczyk R. Performance analysis of extracting object structure from source code / R. Wojszczyk, A. Hapka, T. Królikowski // 27th International Conference on Knowledge Based and Intelligent Information and Engineering Systems (KES 2023) : proceedings. Procedia Computer Science. – 2023. – Vol. 225. – P. 4065–4073. DOI: 10.1016/j.procs.2023.10.402.
13. Method Automated Class Conversion for Composition Implementation / [O. B. Kungurtsev, V. R. Bondar, K. O. Gratilova et al.] // Radio Electronics, Computer Science, Control. – 2024. – № 2. – P. 142–149. DOI: 10.15588/1607-3274-2024-2-14.
14. Shah K. P. Towards Verification of UML Class Models using Formal Specification Methods: A Review [Electronic resource] / K. P. Shah, E. S. Grant // Global Journal of Computer Science and Technology. – 2023. – Vol. 23, № H1. – Regime of access: https://computerresearch.org/index.php/computer/article/view/102296 free (date of the application: 25.10.2025). – Header from the screen.
15. Al-Fedaghi S. Classes in Object-Oriented Modeling (UML): Further Understanding and Abstraction / S. Al-Fedaghi // International Journal of Computer Science and Network Security. – 2021. – Vol. 21, № 5. – P. 139–150. DOI: 10.48550/arXiv.2106.00267.
16. Kungurtsev O. B. Determination of Inheritance Relations and Restructuring of Software Class Models in the Process of Developing Information Systems / O. B. Kungurtsev, A. I. Vytnova // Radio Electronics, Computer Science, Control. – 2022. – № 4. – P. 98–107. DOI: 10.15588/1607-3274-2022-4-8.

DOI 10.15588/1607-3274-2026-2-15