УДК 004.42

АВТОМАТИЗАЦИЯ ПРЕОБРАЗОВАНИЯ ИЗОБРАЖЕНИЙ ДИАГРАММ В ПРОГРАММНЫЙ КОД

А.В. ЛОМАКО, К.А. ХАДЖИНОВА

Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь

Поступила в редакцию 06 апреля 2026

Аннотация. В статье предложен подход к автоматизации преобразования изображений диаграмм в программный код на основе конвейерной обработки. Разработана теоретическая модель, включающая распознавание графических объектов, построение графовой модели, формализацию в терминах конкретной нотации и генерацию кода через унифицированное промежуточное представление. Рассмотрены основные нотации моделирования (UML, BPMN 2.0, сети Петри, IDEF0, IDEF3, DFD, ERD) и их соответствие типам целевого кода. Описан алгоритм обработки изображений диаграмм с выделением этапов предобработки, детекции объектов, построения графа, классификации нотации и кодогенерации. Метод позволяет автоматизировать ручной этап переноса диаграмм в программную реализацию, обеспечивая согласованность модели и кода.

Ключевые слова: диаграммы, автоматизация, распознавание изображений, графовая модель, кодогенерация, UML, BPMN, промежуточное представление, компьютерное зрение, формализация нотаций.

Введение

В программных проектах диаграммы служат не только для визуализации, но и как формальный инструмент описания структуры, поведения и данных системы [1]. В процессе разработки диаграммы вручную переводятся в код, в результате чего графическое представление модели может не полностью совпадать с программной реализацией, занимает определенное время, растет вероятность ошибок и утрачивается исходный смысл модели [2].

Для решения этой проблемы предлагается автоматизация преобразования изображений диаграмм в программный код. В работе предложен последовательный конвейер преобразования изображения диаграммы в программный код, включающий распознавание, построение графовой модели, формализацию в нотацию и генерацию кода.

Обзор существующих решений

В CASE-средствах и системах модельно-ориентированной разработки используются формализованные модели, представленные в форматах UML/XMI, BPMN и во внутренних форматах редакторов [3]. Такие инструменты позволяют автоматически генерировать программный код, SQL-скрипты, конфигурации и другие артефакты, а также поддерживают двустороннюю синхронизацию между моделью и кодом, что сокращает ручной труд и повышает согласованность документации и реализации [4].

Однако большинство решений не рассматривают задачу работы с изображениями диаграмм, полученных в виде скриншотов, отсканированных документов или экспортированных графических файлов. В таком формате диаграмма лишена явной семантики, поэтому ее приходится вручную переносить в формализованную модель, и между изображением и программным кодом остается трудно автоматизируемый ручной этап [5].

Существующие подходы к автоматизации можно разделить на несколько направлений:

– генерация кода из формализованных UML/BPMN-моделей в CASE-средствах без работы с изображениями;

– распознавание графических примитивов и текста на изображениях диаграмм без дальнейшей формализации и генерации кода;

– системы генерации кода для отдельных нотаций (например, ERD, сети Петри) на основе жестких шаблонов без единой теоретической основы [6].

В литературе отсутствует единый подход для преобразования изображений диаграмм в программный код по широкому набору стандартных нотаций [7]. Предлагаемый в работе метод решает данную проблему за счет введения унифицированной теоретической модели и конвейерного алгоритма, позволяющего обрабатывать различные нотации на общей промежуточной и теоретической основе [8].

Классификация диаграмм и формальные основания

Для наглядного представления типов диаграмм и их групп по категориям приведена схема, показанная на рис. 1.

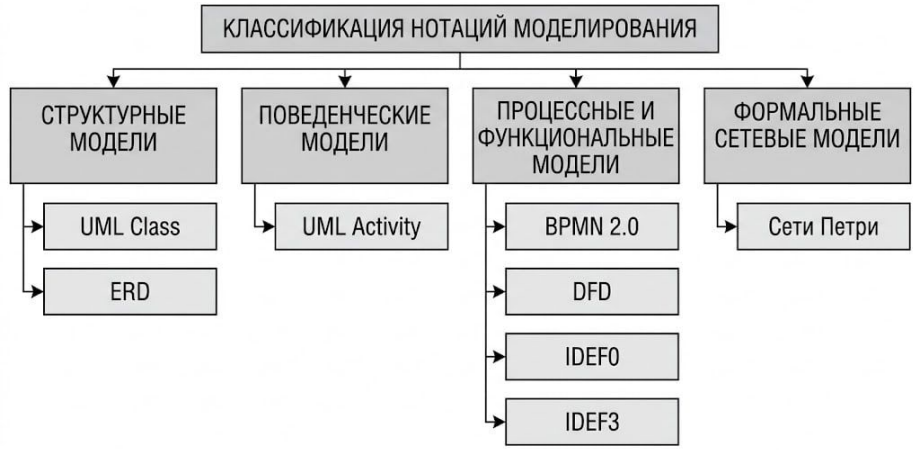

Рис. 1. Иерархия нотаций моделирования

Данная схема иллюстрирует группировку нотаций по типу описания (структура, поведение, процессы, сетевые модели) и по предметной области, показывая, что множество используемых нотаций естественно группируется в несколько типовых категорий. Это позволяет построить единую теоретическую модель преобразования, опирающуюся на единый графовый, и мета-модельный каркас, который унифицирует обработку диаграмм различных типов и позволяет использовать их в рамках одной и той же конвейерной схемы преобразования изображения в программный код.

В рамках предлагаемого подхода рассматриваются несколько основных нотаций, имеющих достаточную формализацию и допускающих прямую трансляцию в программные артефакты [9]. К ним относятся UML (включая диаграммы деятельности и классов), BPMN 2.0, сети Петри, DFD, ERD, IDEF0 и IDEF3 диаграммы [10]. Эти нотации охватывают структурные, поведенческие, процессные и сетевые модели систем, что позволяет использовать их в качестве единой теоретической основы для унифицированного описания и преобразования [11].

Соответствие нотаций, их формального представления и типов целевого кода подробно описано в табл. 1.

Из таблицы 1 видно, что каждая нотация описывается в виде структуры элементов и связей. Несмотря на различия в интерпретации и визуальном оформлении, все рассмотренные нотации можно представить в виде единого графового представления, в котором элементы диаграммы соответствуют вершинам, а зависимости между ними – ребрам этой структуре из теоретической модели, где элементы и связи выражаются через граф G и формальную модель M.

Табл. 1. **Характеристики анализируемых графических моделей**

Нотация	Тип диаграммы	Элементы	Логика	Код/результат
UML Activity	Поведенческая	Начальный узел, действия, переходы, решения (ромбы), слияния, конечный узел	*start → action → if cond → end*	Workflow–скрипт, бизнес–логика *def process_order(order):* *if order.is_valid():* *invoice = create_invoice(order)* *else:* *send_rejection(order)*
UML Class	Структурная	Классы, свойства, операции, связи, наследование, композиция	*class Order; has items[]; relates Customer*	DTO, ORM–модели *class Order:* *def __init__(self, customer_id: int):* *self.customer_id = customer_id* *self.items = []*
BPMN 2.0	Процессная	События начала/окончания, задачи, развилки (исключающие/ параллельные), потоки последовательности	*start → task → exclusiveGateway → end*	BPMN XML, orchestration–код *<process id="OrderProcess">* *<startEvent id="start"/>* *<task id="checkPayment"/>* *<exclusiveGateway id="gateway"/>* *<endEvent id="end"/>* *</process>*
Сети Петри	Формальная сетевая	Места (круги), переходы (прямоугольники), дуги, жетоны (маркировка)	*if marking[p]≥1: fire transition*	Модель верификации *def fire_transition(marking, t):* *if all(marking[p] >= 1 for p in t.inputs):* *for p in t.inputs: marking[p] -= 1* *for p in t.outputs: marking[p] += 1*
IDEF0	Функционально–процессная	Функции (прямоугольники), входы (слева), выходы (справа), управление (сверху), механизмы (снизу)	*decompose process → subproc1, subproc2*	Спецификации процессов *def process_data(input_data):* *validated = validate(input_data)* *transformed = transform(validated)* *return save(transformed)*
IDEF3	Процессная	Процессы, шаги (единицы работ), объекты потоков, узлы связок, альтернативы	*process: step1 → step2 → [alt] step3*	Сценарии исполнения *def etl_pipeline():* *source = extract_from_db()* *if validate(source):* *transformed = transform(source)* *load_to_warehouse(transformed)*
DFD	Поток данных	Процессы (круги), внешние сущности, хранилища данных, потоки данных	*input → process1 → datastore → output*	ETL–логика, архитектура данных *def etl_pipeline():* *raw_data = load_from_source()* *cleaned = clean_data(raw_data)* *save_to_database(cleaned)*
ERD	Структурная модель данных	Сущности (прямоугольники), свойства (овалы), ключи, связи (ромбы), кратность	*Order 1:N OrderItem; Customer 1:1 Order*	SQL DDL, ORM–схема *CREATE TABLE Orders (* *id SERIAL PRIMARY KEY,* *customer_id INT REFERENCES Customers(id),* *total DECIMAL(10,2)* *);*

Теоретическая модель преобразования

Для описания процесса преобразования изображения диаграммы *I* в программный код *C* вводится последовательность отображений, связывающих исходное изображение с конечным кодом через ряд промежуточных представлений. В рамках модели выделяются этапы: анализ

изображения и построение графовой модели, формализация в терминах конкретной нотации и построение промежуточного представления, из которого осуществляется генерация кода.

Общая схема преобразования задается формулой:

$$C = g\left(f_3\left(f_2\left(f_1\left(I\right)\right)\right)\right), \tag{1}$$

где I – исходное изображение диаграммы; f_1 – функция распознавания объектов и построения графа; f_2 – функция формализации, сопоставляющая граф с соответствующей формальной моделью; f_3 – функция построения промежуточного представления; g – функция генерации кода из этого представления.

Формула (1) задает последовательность преобразований изображения в код в рамках единой конвейерной схемы. Исходное изображение рассматривается как двумерная матрица пикселей, что формально описывается в формуле:

$$I = \{p_{ij} \mid i = 1,\ldots,m, j = 1,\ldots,n\}, \tag{2}$$

где p_{ij} – значение яркости или цвета пикселя в позиции (i,j); m и n – высота и ширина изображения в пикселях.

Формула (2) задает базовое представление, с которым работают все этапы предобработки и распознавания [12]. Предобработка включает нормализацию изображения по размеру, контрасту и яркости, а также удаление шумов [13]. Результатом предобработки служит основа для построения множества распознанных объектов, задаваемого формулой:

$$O = \{o_k\}, o_k = \left(b_k, t_k, s_k\right), \tag{3}$$

где b_k – ограничивающий прямоугольник объекта на изображении; t_k – распознанный текст, ассоциированный с объектом; s_k – семантический тип объекта.

Формула (3) задает структуру данных для построения графа. На основе множества объектов O строится ориентированный граф, структура которого задается формулой:

$$G = \left(V, E\right), \tag{4}$$

где V – множество вершин, соответствующих элементам диаграммы; E – множество ребер, отражающих зависимости, потоки данных/управления, переходы или ассоциации между элементами.

Структура каждой вершины графа дополнительно описывается формулой:

$$v_i = \left(\mathrm{type}_i, \mathrm{attr}_i, \mathrm{pos}_i\right), \tag{5}$$

где type_i – тип узла; attr_i – набор атрибутов, относящихся к элементу; pos_i – позиция элемента на изображении.

Формулы (4) и (5) определяют двойную роль графа G: с одной стороны, он хранит топологию связей, с другой – семантические атрибуты элементов диаграммы.

Формализация графа заключается в сопоставлении его элементов с элементами соответствующей формальной модели [14]. Это отображение задается функцией, которая на основе структуры графа и набора типов узлов и ребер определяет корректные элементы и правила модели, как указано в формуле:

$$M = f_2\left(G\right), \tag{6}$$

где M – формальная модель диаграммы в терминах выбранной нотации; f_2 – функция формализации, интерпретирующая граф G в рамках метамодели.

Формула (6) определяет модель *M* как интерпретацию графа *G* в выбранной нотации. Для унификации обработки различных нотаций вводится промежуточное представление, отделенное от этапов распознавания и генерации кода. Его структура в общем виде описывается формулой:

$$IR = \left(N, R, A \right), \tag{7}$$

где *N* – множество узлов, соответствующих элементам предметной области; *R* – множество отношений между ними; *A* – набор атрибутов, содержащих имена, типы, параметры, метки и кратность связей.

Формула (7) задает абстрактный уровень, на котором моделируются любые исследуемые нотации, независимо от их визуальной специфики.

Структура и типовые атрибуты компонентов *IR* подробно описаны в табл. 2.

Табл. 2. **Структура промежуточного представления *IR***

Компонент	Описание	Примеры атрибутов
N – узлы	Элементы предметной области (сущности, задачи, места)	*name, type, subtype, description*
R – отношения	Связи, зависимости, потоки данных/управления	*source, target, type, condition, label*
A – атрибуты	Свойства, значения, метки	*name, value, cardinality, multiplicity*

Использование единого уровня *IR* позволяет унифицировать правила генерации кода для разных нотаций: функция *g* отображает *IR* в целевой код, не завися от конкретной исходной нотации, как указано в формуле:

$$C = g\left(IR \right), \tag{8}$$

где *C* – программа/скрипт/описание в целевом формате; *g* – функция генерации кода, зависящая от целевой нотации и языка программирования.

Формула (8) связывает промежуточное представление с конечным программным кодом.

Общая структура теоретической модели преобразования изображения диаграммы в программный код представлена на схематичной блок-диаграмме, показанной на рис. 2.

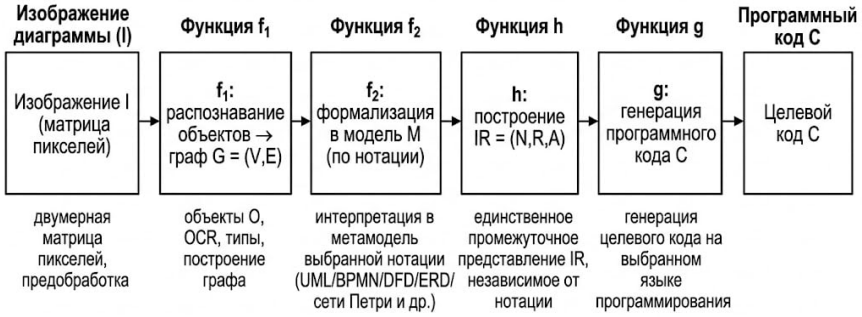

Рис. 2. Модель преобразования «изображение–код»

Теоретическая модель задает единый формальный каркас, в котором распознавание изображения, построение графа, формализация, промежуточное представление и кодогенерация становятся составными частями одной общей схемы, позволяющей унифицировать обработку диаграмм различных типов.

Алгоритм обработки изображения и генерации кода

Алгоритм реализует последовательность преобразований, заданных теоретической моделью, в виде конвейера, в котором изображение диаграммы проходит предобработку, распознавание объектов, построение графа, формализацию, построение промежуточного представления *IR* и генерацию целевого кода.

Общая структура алгоритма представлена на рис. 3 на схематичной блок-диаграмме.

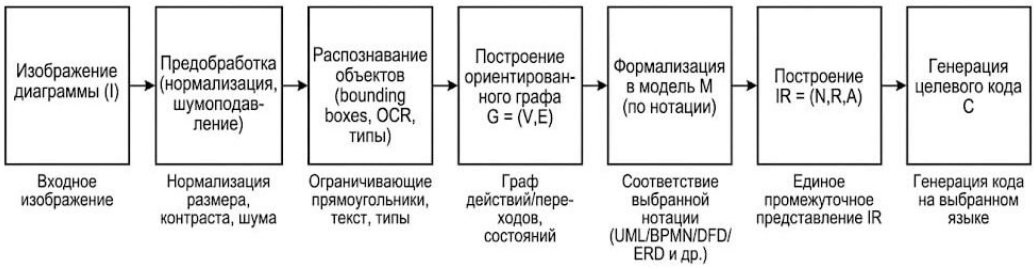

Рис. 3. Схема алгоритма обработки изображения и генерации кода

Рис. 3 иллюстрирует последовательность этапов, в которой выход каждого блока передается на вход следующего: изображение $I \rightarrow$ предобработка $\rightarrow$ распознанные объекты $O \rightarrow$ граф $G \rightarrow$ формальная модель $M \rightarrow$ промежуточное представление IR $\rightarrow$ программа/скрипт C.

Основными промежуточными представлениями, через которые проходят данные, являются распознанные объекты O и граф G. Структура распознанных объектов задается формулой (3), а на их основе строится ориентированный граф G по формуле (4). Структура каждой вершины графа уточняется в формуле (5), что позволяет связать визуальное положение элемента с его семантикой. На следующем шаге граф анализируется для определения типа нотации, и формируется формальная модель M по формуле (6). Эта модель переносится в единое промежуточное представление IR по формуле (7), а из IR генерируется целевой код C по формуле (8).

Структура распознанных объектов и соответствующих вершин графа показана в табл. 3.

Табл. 3. **Семантика вершин графовой модели**

Элемент / тип объекта	Содержимое в O	Содержимое в вершине графа
Состояние / действие	b_k, текст, тип «*state*»	*type*, *name*, *input, output*, *constraints*
Задача / событие	b_k, текст, тип «*task*»	*type*, *name*, *condition, triggers*
Сущность (ERD)	b_k, текст, тип «*entity*»	*type*, *name*, *attributes*, *cardinality*
Переход (сети Петри)	b_k, текст, тип «*transition*»	*type*, *name*, *firing_condition*
Функция (DFD/IDEF)	b_k, текст, тип «*function*»	*type*, *inputs*, *outputs, controls*, *mechanisms*

Данная таблица показывает, как элементы множества O отображаются в вершины графа, сохраняя как визуальное положение, так и семантику, необходимые для последующей классификации нотации и формализации модели.

Алгоритм начинается с предобработки изображения: нормализации размера, контраста и яркости, а также фильтрации шумов. Далее с применением методов компьютерного зрения выделяются графические объекты диаграммы (прямоугольники, окружности, стрелки, текстовые блоки и др.). Для каждого объекта определяется ограничивающий прямоугольник и тип, а также выполняется распознавание текста по области этого прямоугольника [15]. На основе полученных объектов строится ориентированный граф, в котором вершины соответствуют элементам диаграммы, а ребра отражают зависимости, потоки данных/управления или ассоциации. Структура графа анализируется для определения типа нотации, после чего граф формализуется в соответствующую модель M, которая переносится в единое промежуточное представление IR. На завершающем этапе по этому представлению генерируется программный код для выбранной нотации и целевого языка программирования [16].

Алгоритм объединяет теоретическую модель и схему обработки, а также конкретную структуру данных (табл. 2 и 3), в единую схему, позволяющую преобразовывать изображение диаграммы в программный код и адаптировать результат под выбранную нотацию и целевой язык программирования.

Детальную логику алгоритма фиксирует следующий псевдокод:

```
def DiagramToCode(I):
    """
    Преобразование изображения диаграммы в программный код.
    Вход:
        I – изображение диаграммы, матрица пикселей.
    Выход:
        C – программа/скрипт/описание в целевом формате.
    """
    I_pre = preprocess_image(I)
    O_raw = detect_objects(I_pre)
    O = []
    for object_k in O_raw:
        box = bounding_box(object_k)
        text = ocr_single_box(box)
        type_ = classify_object_type(object_k.shape, object_k.position)
        entry = (box, text, type_)
        O.append(entry)
    G = build_graph_from_objects(O)
    notation = classify_notation(G)
    M = formalize_model(G, notation)
    IR = build_intermediate_representation(M)
    C = generate_code(IR, notation, target_language)
    return C
```

Ограничения и перспективы

Качество преобразования диаграммы в код зависит от точности и стабильности распознавания объектов, текста и построения графа. Ошибки, неоднозначная интерпретация примитивов и шумы могут привести к искажению структуры. Метод предполагает стандартный вид диаграммы и соответствие нотациям [17]. При отклонениях (нестандартное оформление, рисование от руки, смешение нотаций) возрастает риск ошибочного сопоставления элементов. IR и функция генерации кода оптимизированы под заданный набор языков и нотаций. Расширение на новые языки потребует доработки правил и шаблонов, увеличивая сопровождение.

В перспективе – интеграция продвинутых моделей компьютерного зрения и языковых моделей для повышения точности [18], расширение поддержки новых нотаций и языков, а также использование IR для анализа свойств моделей (корректность, тупики, избыточные связи) с совмещением кодогенерации и рефакторинга.

Заключение

Предложен подход преобразования изображения диаграммы в код на основе конвейерной обработки: от распознанных объектов и графовой модели до формализации, построения IR и генерации кода. Введена теоретическая модель преобразования через последовательность функций, формализованы структуры данных, графа и IR. Алгоритм реализует конвейер из этапов детекции, распознавания текста, построения графа, классификации нотации, формализации модели и кодогенерации. Архитектура конвейера показана на рисунке 3, структура закреплена в псевдокоде. Унифицированное IR обеспечивает гибкость и масштабируемость подхода для разных нотаций и языков.

Несмотря на ограничения, подход автоматизирует ручной этап преобразования диаграмм в код и может использоваться в инструментах анализа и генерации на основе скриншотов, сканов и экспортированных диаграмм.

AUTOMATION OF DIAGRAM IMAGE CONVERSION TO PROGRAM CODE

A.V. LOMAKO, K.A. KHADZHYNAVA

Abstract. The article proposes an approach to automating the conversion of diagram images into program code based on conveyor processing. A theoretical model has been developed, including recognition of graphical objects, construction of a graph model, formalization in terms of specific notation, and code generation through a unified intermediate representation. The main modeling notations (UML, BPMN 2.0, Petri nets, IDEF0, IDEF3, DFD, ERD) and their correspondence to target code types are considered. An algorithm for processing diagram images is described, highlighting the stages of preprocessing, object detection, graph construction, notation classification, and code generation. The method automates the manual stage of transferring diagrams to program implementation, ensuring model-code consistency.

Keywords: diagrams, automation, image recognition, graph model, code generation, UML, BPMN, intermediate representation, computer vision, notation formalization.

Список литературы

1. Fowler M. UML distilled: a brief guide to the standard object modeling language. 3-е изд. – Reading, MA : Addison-Wesley, 2003. – 209 с.
2. Unified Modeling Language (UML) superstructure. Version 2.5.1 / Object Management Group. – Needham, MA : OMG, 2017. – 768 с.
3. Business Process Model and Notation (BPMN). Version 2.0.2 / Object Management Group. – Needham, MA : OMG, 2018. – 557 с.
4. IEEE Standard Glossary of Software Engineering Terminology : IEEE Std 610.12-1990. – New York : IEEE, 1990. – 84 с.
5. Knuth D. E. The art of computer programming. Vol. 1 : Fundamental algorithms. 3rd ed. – Reading, MA : Addison-Wesley, 1997. – 650 с.
6. Reisig W. Understanding Petri nets: modeling techniques, analysis methods, case studies. – Berlin : Springer, 2013. – 222 с.
7. Gomaa H. Designing software product lines with UML: from use cases to pattern-based software architectures. – Reading, MA : Addison-Wesley, 2004. – 512 с.
8. Sommerville I. Software engineering. 10th ed. – Harlow : Pearson, 2016. – 775 с.
9. Gamma E. Design patterns: elements of reusable object-oriented software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Reading, MA : Addison-Wesley, 1995. – 395 с.
10. Kruchten P. The rational unified process: an introduction. 3rd ed. – Reading, MA : Addison-Wesley, 2004. – 212 с.
11. Satyanarayanan M. Pervasive computing: vision and challenges // IEEE Personal Communications. – 2001. – Vol. 8, № 4. – P. 10–17.
12. Szeliski R. Computer vision: algorithms and applications. 2nd ed. – Springer, 2022. – 950 с.
13. Goodfellow I. Deep learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge, MA : MIT Press, 2016. – 775 с.
14. Куделев А. В. Автоматизация разработки программного обеспечения на основе моделей UML : дис. … канд. техн. наук. – М. : МГТУ им. Н. Э. Баумана, 2023. – 180 с.
15. Иванов П. С. Распознавание структурных элементов блок-схем // Вестник МГТУ. Серия Приборостроение. – 2025. – № 1. – С. 23-34.
16. Петрова Е. А. Генерация SQL-скриптов из ER-диаграмм с использованием компьютерного зрения // Программные продукты и системы. – 2024. – № 2. – С. 56-67.
17. Смирнов А. Б. Моделирование бизнес-процессов в нотации BPMN 2.0. – СПб. : Питер, 2023. – 320 с.
18. Ли Ч. Компьютерное зрение для анализа технической документации / Ч. Ли, М. Ким. – М. : ДМК Пресс, 2024. – 456 с.