https://doi.org/10.25559/SITITO.021.202501.76-89
УДК 519.713

Оригинальная статья

Визуальный графический интерпретируемый язык дискретного моделирования DLAA

Д. А. Гапанович[1*], В. А. Сухомлин[1,2]

[1] ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова», г. Москва, Российская Федерация

Адрес: 119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1

[2] ФГУ «Федеральный исследовательский центр «Информатика и управление» Российской академии наук», г. Москва, Российская Федерация

Адрес: 119333, Российская Федерация, г. Москва, ул. Вавилова, д. 44-22

[*] dim.gapanovich@gmail.com

Аннотация

Статья посвящена разработке визуального графического языка диаграмм состояний и переходов конечных автоматов специального вида (DLAA), предназначенного для структурированного описания конфигурации и поведения дискретно-событийных систем. Приводится анализ известных графических языков, определённых на основе конечных автоматов, таких, как диаграммы Мура, диаграммы состояний Харела, диаграммы конечных автоматов языков SysML и UML, методология автоматного программирования А.А. Шалыто, формализм DEVS. Описан формализм, представляющий собой предметно-ориентированное расширение конечного автомата Мили, на основе которого разработан визуальный графический язык DLAA, ориентированный на спецификацию реактивного поведения системных моделей. Далее рассмотрено расширение конструкций данного языка введением в них семантических функций, называемых, следуя языку SysML, активностями. Реализуемые средствами базового языка программирования активности служит целям погружение языка DLAA в среду программирования базового языка. Описаны особенности реализации языка DLAA как средства, обогащающего базовый язык (в данном случае C#) возможностями спецификации поведенческих аспектов динамических систем. Кратко рассмотрен состав и назначение основных инструментальных средств для автоматизации процесса создания цифровых моделей на основе языка DLAA, включая симулятор, графический редактор, метатранслятор с метаязыка на базовый язык.

Ключевые слова: DLAA, цифровые двойники, Индустрия 4.0, язык графического моделирования, теория автоматов, моделирование дискретных событий, конечные автоматы, автоматное моделирование, кибер-физические системы, диаграммы перехода состояний, системное моделирование, промышленная автоматизация

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Гапанович Д. А., Сухомлин В. А. Визуальный графический интерпретируемый язык дискретного моделирования DLAA // Современные информационные технологии и ИТ-образование. 2025. Т. 21, № 1. С. 76-89. https://doi.org/10.25559/SITITO.021.202501.76-89

© Гапанович Д. А., Сухомлин В. А., 2025

Контент доступен под лицензией Creative Commons Attribution 4.0 License.
The content is available under Creative Commons Attribution 4.0 License.

Original article

Visual Graphical Interpreted Language for Discrete Modeling DLAA

D. A. Gapanovich[a*], V. A. Sukhomlin[a,b]
[a] Lomonosov Moscow State University, Moscow, Russian Federation
Address: 1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation
[b] Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russian Federation
Address: 44 Vavilov St., building 2, Moscow 119333, Russian Federation
[*] dim.gapanovich@gmail.com

Abstract
The article is devoted to the development of a visual graphical language (DLAA) of state diagrams of special-purpose finite state machines (FSM) intended for structured description of configuration and behavior of discrete-event systems. The article provides an analysis of known graphical languages defined on the basis of FSM, such as Moore diagrams, Harel state diagrams, SysML and UML FSM, A.A. Shalyto's automata-based programming methodology, and the DEVS formalism. The article describes a formalism that is a domain-specific extension of the Mealy finite automaton, on the basis of which the DLAA visual graphical language has been developed, aimed at specifying the reactive behavior of system models. Further, the article considers the expansion of the constructs of language DLAA by introducing semantic functions into them, called activities, following the SysML language. Activities implemented by means of the base programming language serve the purpose of immersing the language DLAA into the programming environment of the base language. The article describes the features of the DLAA language implementation as a means of enriching the base language (in this case, C#) with the capabilities of specifying the behavioral aspects of dynamic systems. The composition and purpose of the main tools for automating the process of creating digital models based on the DLAA language are briefly considered. The basic set of such tools includes a simulator, a graphical editor, and a metatranslator from the metalanguage to the base language.

Keywords: DLAA, Digital Twin, Industry 4.0, Graphical Modeling Language, Automata Theory, Discrete Event Modeling, Finite State Machines, Automata Simulation, Cyber-Physical Systems, State Transition Diagrams, System Modeling, Industrial Automation

Conflict of interests: The authors declare no conflict of interest.

For citation: Gapanovich D.A., Sukhomlin V.A. Visual Graphical Interpreted Language for Discrete Modeling DLAA. *Modern Information Technologies and ITEducation.* 2025;21(1):76-89. https://doi.org/10.25559/SITITO.021.202501.76-89

Введение

Современная экономика эпохи Индустрии 4.0 характеризуется всеобъемлющей цифровой трансформацией производств, а именно, массовым переходом промышленных предприятий к технологиям интеллектуального производства, основанного на многомерном высокоточном моделировании свойств и процессов функционирования активов, на роботизированных технологиях и платформах производства продуктов и услуг, на использовании интеллектуальных методов принятия решений для оптимизации и расширения производства.

Одной из центральных парадигм цифровизации производств служит концепция цифровых двойников (ЦД), на основе которой реализуется современный подход к созданию и эксплуатации сложных объектов (физических активов, таких, как: технические системы, производственные процессы, предприятия, цеха, машины и т.п.). Данный подход предполагает использование полномасштабных цифровых моделей физических активов, которые, во-первых, обладают высокой степенью сходства со свойствами и поведением физического актива и, во-вторых, обладают зеркальной информационной взаимосвязью в реальном времени с активом, превращающую последний в целостную кибер-физическую систему *(Cyber-Physical System, CPS)* с возможностью управления ею, как на физическом, так и виртуальном уровнях[1] [1, 2].

Проблема создания множества двойников, применяемых для решения различных задач и в интересах различных участников производства делает актуальной задачу разработки методов и инструментальных средств автоматизации создания ЦД производства [3-5]. При этом к таким методам и средствам во многих случаях предъявляются следующие требования:

1. поддержка современной тенденции «разработки, основанной на моделях», имеющих наглядное графическое представление ЦД, облегчающее их анализ и использование;
2. поддержка модульности ЦД и средств комплексирования двойников активов в двойники систем;
3. поддержка технологической совместимости инструментальных средств с производственным программным обеспечением (ППО), автоматизирующим производственную и организационную деятельности предприятий.

Последнее требование означает, что средства разработки ЦД должны быть включены в базовое программное обеспечение, которое используется для автоматизации производственной деятельности предприятий, что позволяет:

- во-первых, повторно использовать в двойниках функциональные компоненты и информационные ресурсы ППО;
- во-вторых, создавать целостные с технологической точки зрения кибер-физические системы активов и процессов, состоящих из физических (интеллектуальных) активов и их виртуальных образов [6-8];
- в-третьих, облегчить сопровождение всего объёма программного обеспечения.

Именно в этом контексте в работе предпринято исследование возможности разработки математических моделей физических и виртуальных сущностей ЦД, а также протоколов их взаимодействия, на единой дискретно-событийной формальной основе, что позволило разработать математический аппарат для описания моделей ЦД в виде алгебры конечных автоматов специального вида в качестве теоретической базы для построения инструментальных средств автоматизации создания ЦД [9-11].

В данной статье представлены результаты продолжения исследования этой темы в направлении создания инструментальных средств автоматизации построения моделей на основе разработанного математического аппарата, удовлетворяющего рассмотренным выше требованиям. Основным результатов этой работы явилась разработка визуального графического языка диаграмм состояний и переходов конечных автоматов для спецификации конфигураций ЦД и их реактивного поведения и инструментальных средств моделирования.

1. Анализ наиболее известных подходов к описанию поведенческих аспектов систем на основе конечных автоматов

В области разработки визуальных средств спецификации и графических языков для описания поведенческих аспектов систем на основе конечных автоматов имеется значительное число работ. Ниже приведён анализ наиболее известных графических языков спецификации структурных и поведенческих аспектов дискретных управляющих систем на основе теории конечных автоматов.

1.1 Автоматы Мили и Мура

Классическими видами конечных автоматов являются автоматы Мили и Мура[2] [12]. Конфигурация для обоих видов автоматов (с начальным состоянием q_0) одна и та же и имеет следующий вид:

$$M = <A, B, Q, \delta, \lambda, q_0>,$$

где:

A – входной алфавит – непустое конечное множество входных символов (входных событий автомата, например, сигналов от сенсоров),

B – выходной алфавит – непустое конечное множество выходных символов (например, продуцируемых автоматом событий или сигналов),

Q – множество состояний автомата – непустое конечное множество символов состояний автомата (его память),

δ – функции переходов автомата – однозначное отображение вида: $Q \times A \to Q$

λ – функцией выходов автомата – отображение вида: $Q \times A \to B$

q_0 – начальное состояние автомата q_0.

Особенностью автомата Мили является то, что его функция выходов представляет собой отображение вида: Q×A→B, т.е. выход автомата зависит как от текущего состояния автомата,

[1] Прохоров А., Лысачев М. Цифровой двойник. Анализ, тренды, мировой опыт / Под ред. А. И. Боровкова. М.: ООО «АльянсПринт», 2020. 401 с. URL: https://datafinder.ru/files/new4/digital_twin_book.pdf (дата обращения: 26.02.2025); ПНСТ 429-2020 Умное производство. Двойники цифровые производства. Часть 1. Общие положения: предварительный национальный стандарт РФ: издание официальное: утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 7 августа 2020 г. No 38-пнст: введен впервые: дата введения 2021-01-01 / подготовлен АО «ВНИИС», АО «РВК». М.: Стандартинформ, 2020.

[2] Кудрявцев В. Б., Алёшин С. В., Подколзин А. С. Введение в теорию автоматов. М.: Наука, 1985. 320 с.

так и от его текущего входа (входного вектора). В конечных автоматах Мура выходные данные системы, в отличие от автоматов Мили, связаны только с состояниями, т.е. функцией выходов автомата λ является отображением вида: $Q \to B$.

На практике, как правило, автоматы Мили и Мура используются для моделирования управляющей части реактивных систем, логика работы которых определяется возникающими в системе событиями. Выходы автоматов интерпретируется как команды управляемому объекту, а входами автоматов являются как внешние события от окружающей среды, так и внутренние события в системе, например, в виде запросов по обратной связи от управляемого объекта. При реализации программных систем, в которых управляемый объект явно не формализован, функция выходов автомата λ, представляет собой, по существу, переключатель, вызывающий семантические подпрограммы-реакции на входные события.

Автоматы Мили и Мура можно считать эквивалентными, так как показана осуществимость преобразований между конечными автоматами Мура и автоматами Мили и наоборот[3].

Конечные автоматы имеют наглядную форму представления в виде помеченных графов, называемых диаграммами Мура. Такими графами или диаграммами Мура, представляющими конечные автоматы, называются сущности, состоящие из конечного множества вершин-состояний автомата Q, и конечного множества направленных дуг E, соединяющих две вершины (начальную и конечную). В таких графах будут допускаться дуги, имеющие одинаковые начала и концы, называемые также параллельными. Разметка графов осуществляется с помощью отображения $E \to A \times B$, где каждой дуге ставится в соответствие пара – входной символ $a \in A$ и выходной символ $b \in B$. Таким образом, переход из состояния q в состояние q', вызванный входной буквой a, изображается дугой с началом q, концом q' и меткой «a», при этом при переходе по дуге осуществляется вывод выходного символа $b = \lambda(q, a)$.

1.2 Абстрактная модель дискретной системы Уаймора

К наиболее строгим с математической точки зрения подходом использования математической структуры для описания дискретных систем следует отнести исследования Уаймора [13]. В основе подхода Уаймора лежит определение модели дискретной системы, и концепции рецепта связи (*Coupling Recipe*) – аналога блок-схемы системы, связывающей отношениями элементы системы.

В рассматриваемой работе Уаймор определяет модель дискретной системы как кортеж длины пять:

$$Z = \{SZ, IZ, OZ, NZ, RZ\},$$

где:

Z — имя системы

SZ — множество ее состояний

IZ — множество ее входов

OZ — набор ее выходных данных

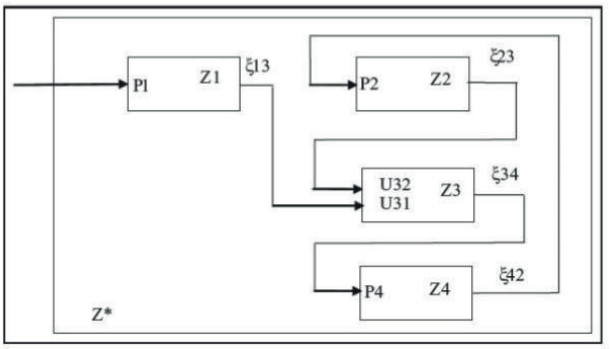
NZ — функция следующего состояния (которая позволяет математически отразить, как входные траектории ведут от одного перехода состояния к другому)

RZ — функция считывания (вывода), которая определяет выходные данные для каждого состояния.

По убеждению Уаймора, данная конструкция совместно с концепцией рецепта связи, обеспечивает возможность для построения модели любой дискретной системы.

Пример, поясняющий применение механизма рецепта связывания [14], исходные данные и постановка задачи примера приведены на рис. 1, где требуется построить результирующую систему $Z^* = RES(K)$ при соединении K систем Zi для $i = \{1\text{-}4\}$ и необходимо указать множество, определяющее связь K. Пример графического изображения входных/выходных связей дискретных систем $Z1, Z2, Z3, Z4$ и как результат связанная система $Z^* = RES(K)$, приведены на рис. 1.

Systems		{Input ports}	Input ports which are assigned for coupling	Output functions
K = {	Z1,	{P1},	Φ, Φ, Φ, Φ,	Φ, Φ, ξ13, Φ,
	Z2,	{P2},	Φ, Φ, Φ, {P2},	Φ, Φ, ξ23, Φ,
	Z3,	{u32, u31},	{u31}, {u32}, Φ, Φ,	Φ, Φ, Φ, ξ34,
	Z4,	{P4},	Φ, Φ, {P4}, Φ,	Φ, ξ42, Φ, Φ }
			Z1 Z2 Z3 Z4	Z1 Z2 Z3 Z4
			Inputs come from	Output connected to



Р и с. 1. Пример связанной системы [14]

F i g 1. Example of a connected system [14]

В спецификации примера используются исходные обозначения Уаймора; для удобочитаемости и ясности добавляются только пробелы в виде буквы Φ внутри таблицы с исходными данными и метки за пределами прямоугольников компонент. В нотации Уаймора спецификация связи состоит из объявления всех моделей компонентов системы и, для каждой модели входных портов, которые назначаются для связи, а также функций вывода и того, к каким моделям компонентов они подключены.

1.3 Автоматы Харела

Д. Харелом разработано расширение традиционного формализма конечных автоматов в виде диаграмм состояний *(Statecharts)*, с целью создания визуального графического языка для описания реактивного поведения дискретных систем, таких как многокомпьютерные системы реального времени, протоколы связи и цифровые блоки управления [15].

Диаграммы состояний Харела расширяют традиционные диаграммы Мура в следующих направлениях:

[3] Hopcroft J., Motwani R., Ullman J. Introduction to Automata Theory, Languages, and Computation. 3rd ed. Pearson, 2006. 560 p.

- структурирование состояний автоматов для поддержки проектирования иерархических системных структур с помощью возможности группирования или кластеризации *(Clustering)* состояний (с исключающим ИЛИ *(XOR-decomposition)*) в сверхсостояние *(Superstate)*;
- введение параллелизма для описания моделей систем с помощью параллельно функционирующих групп состояний *(И-декомпозиции – AND-decomposition)*; названного свойством ортогональности *(orthogonality)*, с некоторыми возможностями синхронизации;
- расширение средств связи *(communication)* между автоматами, включая широковещательную связь;
- поддержка возможности уточнения *(refinement)* состояний;
- поддержка истории работы системы в группе состояний для возможности при повторном входе в группу состояний входить в последнее перед выходом исполняемое состояние, при этом данный механизм пролонгируется на все уровни иерархической системы состояний;
- возможности использования временных ограничений и таймаутов с применением неявных таймеров.

1.4 Автоматное программирование А.А. Шалыто

Близкий по возможностям к методу диаграмм состояний *(Statecharts)* Харела подход, названный автоматным программированием, разработан профессором А.А. Шалыто. Более точное название этого подхода «программирование с явным выделением состояний»[4]. Подход Шалыто ориентирован на класс систем, названных автором системами со сложным поведением, который является более широким, чем класс реактивных систем в подходе Харела.

Особенность данного подхода состоит в том, что моделируемые системы представляются в виде автоматизированных систем управления (или приводятся к такой структуре), состоящих из управляющего автомата и управляемого объекта, которые связаны прямой и обратной связями. Прямая связь используется для передачи команд от управляющего автомата управляемому объекту, а обратная – для передачи запросов от объекта к автомату, поступающих в автомат в виде входных переменных (наряду с внешними событиями от внешней среды). Базовым формализмом для управляющего автомата являются автоматы Мура и Мили. В случае моделирования сложных систем они представляются в виде структурированного множества взаимодействующих автоматов, совместно управляющих множеством объектов управления.

К достоинствам программировании с явным выделением состояний можно отнести следующее:

- автоматное программирование основывается на классических автоматных моделях (автоматах Мили и Мура) и вносит только два основных расширения (вложенные и вызываемые автоматы), что делает нотацию диаграмм переходов в автоматном программировании более простой по сравнению с языком Statecharts, но при этом позволяет описывать те же свойства в поведении систем;
- применение методов автоматического синтеза управляющих автоматов по заданным функциям перехода и выхода с помощью кодирования в двоичном коде состояний, входных и выходных алфавитов автоматов;
- разработка для данного подхода инструментальных средств генерации кода на языке С по графам переходов[5] [16];
- разработка данного подхода как для функциональной, так и объектно-ориентированной парадигм программирования.

1.5 Конечные автоматы в языках SysML и UML

В языке UML/SysML реализован вариант конечных автоматов *(Finite State Machine)* в виде диаграмм машин состояний *(State Machine Diagram)*, пожалуй, наиболее насыщенный средствами, расширяющими описательные возможности для спецификации поведенческих аспектов систем, по сравнению с классическими конечными автоматами[6] [17-19].

К числу таких расширений следует отнести:

1) семантические активности, связанные с состояниями автомата [20], включая:
- активность, которая выполняется при входе в состояние *(оператор entry);*
- активность, выполняемая в самом состоянии *(оператор do);*
- активность, выполняемая при выходе из состояния *(оператор exit).*

2) активность, связанная с переходами между состояниями, которые связываются с дугами на диаграмме состояний;

3) возможность одновременного использования как активностей, связанных с состояниями, так и активностей, связанных с переходами;

4) определение начального состояния автомата, с которого начинает работу автомат при передаче ему управления, также начальное состояние определяется и в супер-состояниях, в случае иерархической декомпозиции состояний (начальное состояние обозначается в диаграмме закрашенным кругом);

5) определение конечных состояний в диаграмме состояний, т.е. состояний при достижении которых автомат завершает свою работу и, в случае вызова внешним автоматом, возвращает ему управление, включая выход по стеку вызовов из других объемлющих конечных автоматов (конечное состояние обозначается в диаграмме кругом с закрашенным кругом внутри);

6) супер-состояния, представляющие собой конечные наборы состояний и вложенных супер-состояний, что поддерживает иерархический подход в разработке моделей систем, позволяет существенно сократить число переходов (дуг в диаграмме), упростив описание сложного поведения;

7) ортогональность – параллельность исполнения независимых областей состояний (супер-состояний);

8) контроль событий, связанных с переходом состояний с по-

[4] Поликарпова Н. И., Шалыто А. А. Автоматное программирование. 2-е изд. СПб: Питер, 2020. 176 с. URL: https://ibooks.ru/bookshelf/26248/reading (дата обращения: 26.02.2025).

[5] Головешин А. Использование конвертора Visio2Switch [Электронный ресурс] // Университет ИТМО, 2025. URL: http://is.ifmo.ru/progeny/visio2switch (дата обращения: 26.02.2025).

[6] Systems Modeling Language (OMG SysML) : сайт [Электронный ресурс] // Object Management Group, 2025. URL: https://www.omg.org/sysml/ (дата обращения: 26.02.2025).

мощью сторожевых предикатов защиты *(Guards)*, которые в зависимости от значения условий, разрешают или блокируют переход;

9) запоминание истории выполнения диаграмм для обеспечения возможности при повторном входе в диаграмму продолжить ее выполнение с последнего исполнявшегося при выходе из нее состояния;

10) введение задержек и тайм-аутов для отражения в модели временных аспектов;

11) введение ассоциаций с другими диаграммами состояний, в том числе с другими элементами модели и некоторые другие средства, расширяющие классические конечные автоматы.

1.6 DEVS – формализм для моделирования сложных динамических систем с использованием абстракции дискретных событий

DEVS *(Discrete-Event Modelling and Simulation)* – наиболее мощный формализм для моделирования сложных динамических систем с использованием абстракции дискретных событий, созданный профессором Бернардом Зейглером и его школой [21-24]. Вообще, формализм DEVS не ограничивается только моделированием дискретных систем, а охватывает и другие виды моделирования, включая: непрерывное, дискретное, параллельное, сетевое и марковское (вероятностное). В работе [21] показано, что все виды моделирования могут быть реализованы на основе дискретно-событийного подхода.

Здесь мы проанализируем возможности классического DEVS, предназначенного для моделирования дискретных событийных систем.

Формализм DEVS определяет два типа моделей:

- атомарные модели, обеспечивающие спецификации динамики компонентов системы;
- связанные модели, которые описывают, как соединить несколько компонентов модели (которые могут быть атомарными или связанными) вместе, чтобы сформировать новую модель.

Иерархический способ построения моделей основан на доказательстве замкнутости операции связывания *(coupling)* формализма DEVS.

Атомарную модель DEVS можно рассматривать как автомат с набором состояний и функциями перехода, изменяющими состояние при возникновении внешнего события или по истечении времени пребывания в состоянии. Когда внешних событий не происходит, состояние атомарной модели обновляется внутренней функцией перехода по истечении срока жизни состояния. Когда происходит внешнее событие, атомарная модель изменяет своё состояние, применяя свою внешнюю функцию перехода. Время жизни состояния определяется функцией продвижения времени *Ta (time advance function)*. Каждое изменение состояния может создавать выходные сообщения через функцию вывода.

Атомарные модели являются неделимыми строительными блоками модели.

Атомарные модели, описываются следующей конфигурацией:

$$<X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, Ta>,$$

X – набор внешних событий (входов в систему)

Y – набор выходов

S – набор состояний модели

q_{init} -начальное общее состояние, включающего как само начальное состояние (s_{init}), так и прошедшее время **e** (**Elapsed time**), определяющее как долго система находится в этом состоянии (например, учитывая результат завершения предыдущего сеанса моделирования)

$\delta_{int}: S \to S$ – функция внутреннего перехода, $\boldsymbol{\delta_{int}}$ определяет следующее состояние для каждого состояния, если оно не конечно, каждое состояние имеет не более одного следующего состояния, что предотвращает недетерминизм

$\delta_{ext}: SxX \to S$ – внешняя функция перехода, которая определяет новое состояние системы при возникновении внешнего события в зависимости от текущего состояния

$\lambda: S \to Y \cup \{\varphi\}$ – выходная функция, реализующая генерацию выходных событий. Следует учитывать, что событие генерируется перед выполнением внутренней функции перехода, т.е. до смены состояние. Функция вывода может не возвращать никаких результатов, и в этом случае она возвращает специальный символ φ.

$Ta: S \to R$ – функция продвижения времени *(Ta) (Time Advance)*. Для каждого состояния определена функция, которая возвращает длительность нахождения модели в соответствующем состоянии. Длительность может быть любым неотрицательным действительным числом, включая бесконечность.

Под связанной моделью в DEVS понимается структурная модель, объединяющая как атомарные модели, так и связанные подмодели. В отличие от атомарных моделей, связанная модель не определяет собственно поведенческих аспектов систем. Она отвечает за структурную организацию модели.

Для определения базовой структуры связной модели используются три элемента.

1) Экземпляры модели (D)

Набор экземпляров атомарных моделей определяет, какие модели включены в связанную модель:

$$\boldsymbol{MS = \{M_i \,|i \in D\}}$$

2) Спецификации экземпляров модели (MS),

где M_i – спецификация i-ой атомарной модели. Для каждого элемента, определенного в D, спецификация представляет восьмиэлементный кортеж, определяющий атомарную модель.

$$\boldsymbol{MS = \{M_i \,|i \in D\} = \{<X_i, Y_i, S_i, q_{init,i}, \delta_{int,i}, \delta_{ext,i}, \lambda_i, ta_i> \,|i \in D\}}$$

Формальная семантика исполнения моделей в DEVF определяется с помощью связанного с формализмом DEVS алгоритма абстрактного симулятора *(abstract simulator)*, обеспечивающего корректную реализацию поведения моделей.

По определению, подмодель связанной модели DEVS всегда должна быть атомарной моделью (в принципе, это расширяется для поддержки произвольных иерархий).

3) Влияние модели *(Model influencees)* – $(IS = \{I_i | i \in D \cup \{self\}\})$

Помимо определения экземпляров модели и их спецификаций, необходимо определить связи между ними. Связи определяются с помощью наборов влияний (**influencee sets**): для каждого экземпляра атомарной модели определяется набор моделей, на которые влияет эта модель. Существует ограничение на связи, чтобы гарантировать невозможность создания несогласованных моделей: модель не должна влиять сама на себя, т.е. модель не должна быть элементом собственного набора влияний. $\forall i \in D : i < I_i$. Разрешены только связи внутри связанной модели, т.е. модели не должны напрямую влиять на модели за пределами текущей связанной модели, а также

на модели, находящиеся глубже внутри других подмоделей на этом уровне [25].

Как следствие, связанную модель можно определить с помощью трех элементного кортежа:

$$<D, MS, IS>$$

Однако, чтобы связанная модель могла взаимодействовать с внешним миром, она дополняется собственными входными и выходными событиями, которые служат интерфейсом для связанной модели. Тогда к этому кортежу добавляются компоненты *Xself* и *Yself*, соответственно набор входных и выходных событий, в результате чего получается кортеж из 5 элементов:

$$<X_{self}, Y_{self}, D, MS, IS>$$

Отметим ещё два механизма классического DEVS:

4) Это функция разрешения конфликтов (**select**), которая используется при выполнении сложных многокомпонентных моделей, когда возникает неопределённость в однозначном выборе порядка исполнения компонент модели. Эта функция принимает все конфликтующие модели и возвращает ту, которая имеет приоритет над остальными, решая конфликт эвристическими методами.

select: $2^D \rightarrow D$

5) Функция трансляции (ZS), которая позволяет переименовывать выходные события моделей так, чтобы согласовать выходы со входами в другие модели. Функции трансляции определяются для каждого соединения, в том числе между входными и выходными событиями связанной модели.

$$ZS = \{Zi,j \mid i \in D \cup \{self\}, j \in Ii\}$$

Функция трансляции неявно считается тождественной функцией, если она не определена. В итоге связанная модель описывается семиэлементной конфигурацией элементов следующего вида:

$$<X_{self}, Y_{self}, D, MS, IS, ZS, select>$$

Проведённый выше анализ наиболее известных средств описания поведенческих аспектов систем на основе конечных автоматов показал, что:

- графические языки на основе конечных автоматов широко и успешно применяются в моделировании поведенческих аспектов дискретных динамических систем,
- реализации графических языков представляют собой, как правило, замкнутые целостные для реализации технологии, которые сложно совмещать с программным заделом, используемым для автоматизации производства.

В связи с чем, в статьи описана разработка нового графического языка, названного DLAA, характеризующегося компактным и, одновременно, полным набором базовых средств для описания поведенческих аспектов динамических систем. Кроме этого данный язык прост в реализации и поддерживает требование реализации как инструмента обогащения базовых средств разработки программного обеспечения автоматизации производства.

2. Формализм, основные понятия и возможности языка DLAA базового уровня

Визуальный графический интерпретируемый язык дискретного моделирования DLAA представляет собой язык диаграмм состояний и переходов конечных автоматов специального вида, с помощью которых описывается динамика поведения исследуемых дискретно-событийных систем. Поэтому сначала рассмотрим формализм спецификации конечного автомата, лежащего в основе языка DLAA. Определяемый с помощью данного формализма класс автоматов, можно считать предметно-ориентированным расширением автомата Мили.

2.1 Формализм спецификации конечного автомата

Для целей дискретно-событийного моделирования систем понятие конечного автомата Мили *(FSM – Finite-state machine)*, взятого за основу в работе [9] алгебры конечных автоматов, расширим до спецификации, имеющей следующее представление:

$$FSM = <Ex, Ey, E_I, S, F, q_{init}, \delta_{int}, \delta_{ext}, \lambda_{st}, \lambda_{tr}, Ta, TS>, \text{ где} \quad (1)$$

Ex – множество внешних входящих событий автомата

Ey – множество внешних исходящих событий автомата

$E_I = \{i, *, **\}$ – множество внутренних событий автомата

S – множество состояний автомата

F – множество конечных состояний автомата

q_{init} – начальное состояние автомата

δ_{int} – функция внутреннего перехода

δ_{ext} – функция внешнего перехода

λ_{st} – функция вывода по состоянию

λ_{tr} – функция вывода по переходу

Ta – функция длительности пребывания в состоянии

TS – структура времени автомата

К характерным особенностям такого расширения понятия автомата Мили относятся:

1) Разделение множества событий в автомате на внутренние (обусловленные заданной логикой автомата) и внешние *(E)*.

2) Различение внешних событий перехода *E* для каждого автомата на входящие *(Ex)* и исходящие *(Ey)*.

3) Введение следующих внутренних событий автомата:

i – событие инициализации автомата, с помощью которого автомат становится активным и устанавливается в начальное состояние

* – событие входа в состояние, рассматриваемого как начало выполнения некоторого (производственного) процесса

** – событие выхода из состояния, моделирующее выход из процесса состояния.

4) Для каждого автомата вводится функция внутреннего перехода (δ_{int}) из одного состояния в другое:

$\delta_{int}: S \rightarrow S$, где *S* – множество состояний автомата.

Функция внутреннего перехода для каждого состояния определяет следующее состояние производственного цикла, описываемого автоматом. В отличие от DEVS имеется возможность перехода из одного состояния в несколько различных состояний в зависимости от значения управляющего предиката для каждого перехода, определяемого динамически. В некоторых состояниях следующее состояние может отсутствовать. Такие состояния считаются конечными для автомата. Они принадлежат множеству *F*.

5) Для каждого автомата вводится функция внешнего перехода (δ_{ext}):

$\delta_{ext}: S \times Ex \rightarrow S$

Функция внешнего перехода определяет новое состояние автомата в зависимости от текущего состояния и внешнего входящего события. Для внешних входящих событий функция внешнего перехода имеет приоритет перед событием внутреннего перехода.

6) Для каждого автомата определена функция длительности состояния *(ta) (Time Advance)* – аналогичная функции продвижения времени в системе DEVS [21], ставящая в соответствие длительность нахождения модели в соответствующем состоянии. Длительность может быть любым действительным неотрицательным числом (в том числе случайным числом), включая бесконечность. Длительность равная нулю, как правило, используется в искусственных состояниях.

7) В отличие от других автоматных технологий в автоматах рассматриваемого вида используются две функции вывода – функция вывода по состоянию (λ_{st}) и функция вывода по переходу (λ_{tr}). Использование обеих функций вывода, в частности, применяется в режиме отладки моделей.

Функция λ_{st} реализует (если определена) генерацию исходящих сообщений в буфер вывода, когда автомат переходит в конкретное состояние. Вывод может содержать внешнее исходящее событие (из *Ey*), поэтому функция срабатывает перед выполнением внутренней функции перехода. В случае, если функция для некоторого состояний не определена, считается, что функция генерирует пустое сообщение φ.

$\lambda_{st}: S \rightarrow Ey \cup \{\varphi\}$

Функция λ_{tr} реализует (если определена) свой вывод аналогично функции λ_{st}, но срабатывает, когда автомат, находясь в состоянии *S*, получает на входе входящее событие из множества *E* или внутреннее событие выхода из состояния **, после чего осуществляется переход в новое состояние.

$\lambda_{tr}: SxEx \rightarrow Ey \cup \{\varphi\}$

8) Понятие состояния данного класса автоматов рассматривается как абстракция некоторого производственного процесса, в общем случае имеющего длительность во времени. Длительности равные нулю или бесконечности также возможны как частные случаи, что часто используется в технических решениях при моделировании систем. Моделирование процесса, протекающего в конкретном состоянии автомата реализуется с помощью рассмотренных выше двух типов внутренних событий – входа в состояние, рассматриваемого как начало производственного процесса, и выхода из состояния, моделирующего выход из процесса. Внутренние события входа в состояние обозначаются звёздочкой – *, а события выхода из состояния – двумя звёздочками – **.

9) В общем случае каждый автомат обладает собственной структурой времени $\boldsymbol{TS_A}$. Поэтому при моделировании систем, состоящих из нескольких компонент, представленных в модели отдельными автоматами, работающими в собственных временных структурах, важной задачей становится приведение временных структур компонент к общей временной структуре системы. Примером здесь может служить модель центрального процессора, компоненты которого такие как, арифметическое и логическое устройство, память, кэши работаю в собственных тактовых режимах.

В дальнейшем описанный класс конечных автоматов будем называть автоматами **Мили***.

2.2 Формализм спецификации системы конечных автоматов

Рассмотренный в предыдущем разделе формализм обладает необходимыми возможностями для описания поведения автоматных событийных моделей отдельных устройств. Для применения разработанного на основе данного формализма языка применительно к моделированию многокомпонентных систем введём формализм, представляющий модель системы, в виде следующей конфигурации:

$$\boldsymbol{SM=<N, E, I, Q, \Delta, \Lambda, D, SPM, Select, TS\_SM>}, \quad (2)$$

где

N – имя модели системы *SM (System_Model)*

E – множество событий системы – объединение всех внешних событий (входящих и выходящих) компонентов-автоматов

I = {i, ...} – множество внутренних системных событий, обязательным является событие *i*, рассылаемое системой при инициализации автоматов из множества *D* (перевод их в начальное состояние и активный режим работы)

Q – множество состояний системы представляется в виде прямого произведения состояний состояний компонентов, т.е.

$$\boldsymbol{Q = S_1 x S_2 x ... S_{|D|}}$$

Δ – обобщённая функция перехода состояний в системе при наступлении внешних и внутренних событий, распространяется на все компонеты-автоматы системы

Λ – обобщённая функция вывода при наступлении внешних и внутренних событий в системе

$D = \{d_i\}$ - $i \in [1 – k]$, где *k* – число автоматов *(FSM)* в системе, множество ссылок на автоматы, входящие в состав модели

$SPM = \{SP\_FSM_{di}\}$, где SP_FSM_{di} – спецификация автомата $d_i \in D$ – множество спецификаций автоматов системы

Select – функция разрешения конфликтов в случае сингулярного планирования последовательностей событий в системе, т.е. событий, относящихся к одному моменту времени, но упорядоченных по выполнению

TS_SM – структура времени системы.

Определённая формулой (2) модель многокомпонентной системы предполагает параллельное/квазипараллельное исполнения моделей компонентов (автоматов или FSM_{di}), при этом предполагается, что структуры времени компонентов приведены к структуре времени модели системы.

2.3 Визуальный графический интерпретируемый язык дискретного моделирования DLAA (базовый уровень)

Язык DLAA предназначен для описания конфигурации автоматов Мили* и имеет две формы представления: текстовую и графическую.

Текстовая форма записи конфигураций автоматов основана на последовательном обходе элементов конфигурации автомата (2) и записью определений этих элементов в традиционной теоретико-множественной нотации.

Графическая форма чрезвычайно проста. Она включает три основных элемента:

1) графическое представление состояния в виде прямоугольника с закругленными углами (аналогично тому, как принято в языке SysML), внутри которого помещается имя состояния;
2) переход в диаграмме из одного состояния в другое в виде стрелки;
3) набор разметок переходов:

- для перехода по внешнему событию над стрелкой помещается имя этого события, если есть вывод по состоянию и/или переходу, то выводы предваряются восклицательным знаком и записываются за именем события или, что предпочтительнее, под чертой, как знаменатель дроби;
- для перехода по внутреннему событию имя события не используется, а выводы отображаются аналогично случаю перехода по внешнему событию;

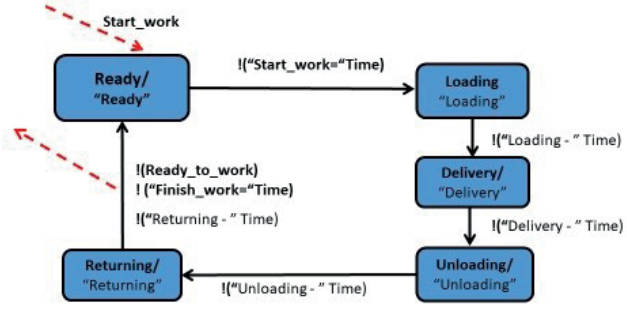
- также с целью повышения наглядности диаграмм могут использоваться пунктирные стрелки, показывающие применение входящих событий и исходящих событий.

Проиллюстрируем графическую форму записи автоматов на следующем примере.

Выводимая информация с помощью функции вывода на диаграмме будет записываться в скобках, перед которыми ставится восклицательный знак:

- выводимая текстовая информация заключается в кавычки
- вместо имени переменной выводится ее значение
- сообщение об исходящем событии заключается в скобки, предваряемые восклицательным знаком.

Рассмотрим в качестве примера модель работы полуавтоматического погрузчика, рис. 2. Погрузчик запускается оператором по команде (для автомата входящему внешнему событию – *"Start_work"*). Начальное состояние автомата q_{init} есть *Ready*. При переходе из состояния готовности *(Ready)* в состояние начала производственного цикла *(Loading)* с помощью функции вывода λ_{tr} генерирует вывод времени начала цикла работы погрузчика *(«Start_work-"Time)*. Далее погрузчик, проходя последовательно по технологической цепочке состояний *(Loading, Delivery, Unloading, Returning)*, выполняет следующие процедуры: загрузку руды, доставку руды к месту выгрузки, выгрузку руды, возврат на рабочее место. При этом после возврата на рабочее место, при переходе вновь в состояние готовности, с помощью функции вывода λ_{tr} генерирует вывод оператору о готовности к работе *«!(Ready_to_work)»*. Также этой же функцией выводится время завершение полного цикла работы *(«Finish_work-"Time)*.



Р и с. 2. Пример диаграммы на языке DLAA0 с описанием работы полуавтоматического погрузчика

F i g. 2. An example of a diagram in DLAA0 language describing the operation of a semi-automatic loader

Источник: здесь и далее в статье все рисунки составлены авторами.

Source: Hereinafter in this article all figures were drawn up by the authors.

2.4 Расширение конструкций базового языка DLAA функциональной семантикой

Рассмотренные в предыдущем разделе язык DLAA обладает достаточно адекватными возможностями для моделирования поведения дискретных событийных систем. Ещё одной особенностью языка DLAA является то, что он разработан как средство, обогащающее базовый язык программирования (в рассматриваемой работе это C#), чтобы создать эффективный инструмент для построения моделей систем указанного класса. Для достижения этого необходимо решить две задачи:

1) связать элементы интерпретируемого языка DLAA0 с программными элементами базового языка программирования;
2) реализовать погружение языка в среду программирования базового языка.

Рассмотрим решение первой задачи, а именно, включение в диаграммы DLAA программных элементов, написанных на базовом языке, которые дополняют логику событийного управления поведением модели функциями семантической обработки информации, реализуемой средствами базового языка.

Такое развитие языка DLAA, и соответствующее развитие понятий автомата Мили* будем называть расширением языка и конструкций автомата функциональной семантикой.

Определение. Фрагменты программного кода, связанные с событиями и переходами и предназначенные для выполнения семантических действий, будем называть **активностями**.

Активности подразделяются на:

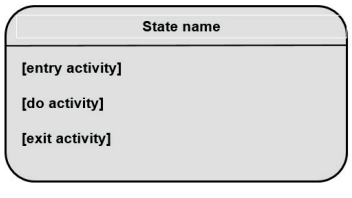
1) **активности состояния** – программы, выполнение которых связано с состоянием автомата (активности типа **entry**, активности типа **do**, активности типа **exit**);
2) **активности переходов** (выполняются в случае перехода из одного состояния в другое, могут контролироваться сторожевым предикатом);
3) **активности рекурсивного вызова** – активности типа **pref** и активности типа **suf** (рассмотрены ниже).

Для управления выполнением активностей переходов могут использоваться сторожевые предикаты на переходе, которые разрешают переход только в случае истинного значения предиката. Выбор истинного предиката осуществляется однонаправленным сканированием сторожевых предикатов ветви переходов. В случае истинности нескольких таких предикатов, так и в случае отсутствия истинных предикатов, диаграмма автомата считается некорректной.

Активности состояния конечного автомата, как и языке SysML [17], могут быть следующих типов[7]:

- входная активность *(entry)* – если определена, выполняется при входе в состояние автомата;
- рабочая активность *(do)* – если определена, выполняет основную работу, совершаемую автоматом в данном состоянии;
- выходная активность *(exit)* – если определена, выполняется непосредственно перед переходом из состояния.

Типы активностей состояния конечного автомата показаны на рис. 3.



Р и с. 3. Типы активностей состояния конечного автомата

F i g. 3. Finite State Machine Activity Types

[7] Там же.

Соответствие между состояниями автоматов и их активностями задаётся с помощью функции:

$$\boldsymbol{\mu}: S \rightarrow \langle A_{entry}, A_{do}, A_{exit} \rangle|,$$

где

A_{entry} – множество активностей типа *entry*,
A_{do} – множество активностей типа *do*,
A_{exit} – множество активностей типа *exit*,

Соответствие между переходами автоматов и активностями задаётся с помощью отображения:

$$\boldsymbol{\text{Ŋ}}: (S, x) \rightarrow (A_{tr}),$$

где

A_{tr} – множество активностей перехода

Отсутствующие элементы отображений **μ** и **Ŋ** будем заменяются пробелами.

Тогда полное описание автомата в языке DLAA1 будет иметь вид кортежа длины 14:

$$\boldsymbol{FSM = \langle Ex, Ey, E_f, S, F, q_{init}, \delta_{int}, \delta_{ext}, \lambda_{st}, \lambda_{tr}, Ta, TS_A, \mu, \text{Ŋ} \rangle},$$ где (3)

Как рассматривалось выше, переход в следующее состояние автомата может выполняться или с помощью функция внешнего перехода (δ_{exit}) как реакция на внешнее событие, или, если внешнего события на входе автомата не появилось, то с помощью функция внутреннего перехода (δ_{int}), определяющую регулярную последовательность выполнения работы автоматом.

Заметим, что, если в базовом варианте языка DEVS для каждого состояние определяется не более одного последующего состояния, то в языке DLAA возможны несколько ветвлений, каждое из которых управляется предшествующим переходу сторожевым предикатом. Естественным ограничением для такой конструкции является равенство истине только одного и только из предикатов такой ветви переходов.

С переходом в новое состояние может быть также связана активность вместе с возможным сторожевым предикатом.

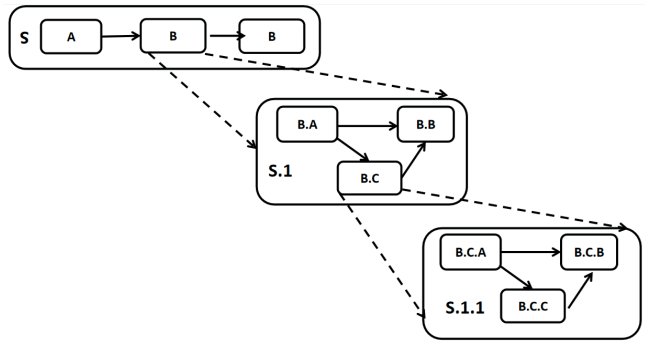
Важно отметить, что, как и в языке SysML, будет допускаться возможность одновременного использования в языке DLAA как активностей, связанных с состояниями, так и активностей, связанных с переходами.

События

События могут иметь список параметров. При наступлении события в параметрах обрабатывающей событие программе перелается контекстная информация. Механизм передачи параметров реализуется по правилам, определяемым в базовом языке. Механизм возбуждения события и передачи параметров в обрабатывающую событие программу осуществляется с помощью специального объекта, называемого уведомлением о событии, определяемого ниже.

2.5 Структурирование состояний

Разработка моделей сложных систем, как правило, осуществляется поэтапно, начиная с самого общего укрупненного представления системы, и затем последовательно от этапа к этапу повышая уровень детализации системного представления. В связи с чем для поддержки иерархического подхода построения моделей систем, предполагающего возможность перехода на каждом этапе проектирования к все боле детальному представлению моделей систем, в язык DLAA введены средства структурирования состояний диаграмм, позволяющие представлять их на более детальном уровне в виде самостоятельных диаграмм. Такой иерархический способ проектирования моделей систем можно сравнить с суперсостояниями в языке Харела [15] или языках SysML/UML [17]. Данный подход иллюстрируется на рис. 4, на котором показаны три уровня детализации представления системы S: на верхнем уровне система представляется как композиция трёх состояний/процессов A, B, C; на следующем уровне состояние системы B представляется в виде подсистемы S.1 с состояниями B.A, B.B, B.C; и на последнем уровне детализации состояние B.C представляется подсистемой S.1.1.



Р и с. 4. Иерархический подход к проектированию моделей системы с детализацией элементов модели при переходе на уровень более детального ее представления

F i g. 4. A hierarchical approach to designing system models with detailing of model elements when moving to a more detailed level of its representation

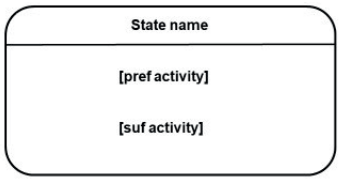
В языке DLAA для реализации механизма структурирования диаграмм используется введенная в алгебре DTA *(Digital Twin Algebra)* [9] операция рекурсивного вызова автомата. Введение новой возможности в язык не потребовало разработки дополнительных конструкций. Просто имена автоматов в таком случае интерпретируются как имена составных состояний и вход в такое состояние по внутреннему (*) или внешнему событию *(x)* вызывает переход в начальное состояние вызываемого автомата, а в стеке возвратов Return_stack вызывающего автомата сохраняется точка вызова в диаграмме для возврата управления из вызываемого автомата после того, как тот завершит своё конечное состояние. Конечным состоянием считается такое, для которого отсутствует последующее состояние и отсутствует переход по внешнему событию.

В случае использования вложенных автоматов области видимости определяемых в них сущностей (состояний, событий) формируются на традиционном для языков программирования с блочной структурой принципе – в область видимости каждого автомата входят сущности, определённые в данном автомате и объемлющих.

Далее, следуя подходу расширения языка функциональной семантики, для состояния рекурсивного вызова вводится два вида активностей:

A_{pref} – префиксная активность
A_{suf} – и суффиксная активность

которые трактуются как префиксная и суффиксная семантические функции, выполняемые до рекурсивного вызова автомата и после возврата из него соответственно. Общий вид состояния рекурсивного вызова показано на рис. 5.



Р и с. 5. Общий вид состояния-рекурсивного вызова
F i g. 5. General view of the recursive call state

Соответствие между состояниями автоматов и активностями, которое задаётся с помощью отображения **μ,** теперь будет иметь следующий вид:

$$\boldsymbol{\mu}: S \rightarrow \langle A_{entry}, A_{do}, A_{exit} \rangle | \langle A_{pref}, A_{suf} \rangle,$$

В качестве примера применения операции рекурсивного вызова автоматов рассмотрим событийную модель работы грузовика на рис. 6, где показано, что грузовик, выполняя традиционный рабочий процесс, проходит следующую последовательность состояний:

InitTruck – начальное состояние

LoadingTruck – состояние, в котором выполняется загрузка руды в грузовик Truck

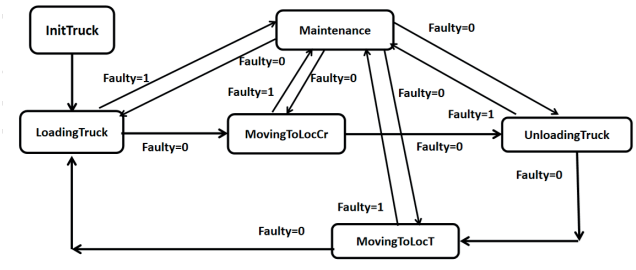
MovingToLocCr – состояние, в котором осуществляется перемещение грузовика Truck в локацию для выгрузки

UnloadingTruck – состояние, в котором выполняется выгрузка руды из грузовика

MovingToLocT – состояние, в котором осуществляется перемещение грузовика к месту загрузки

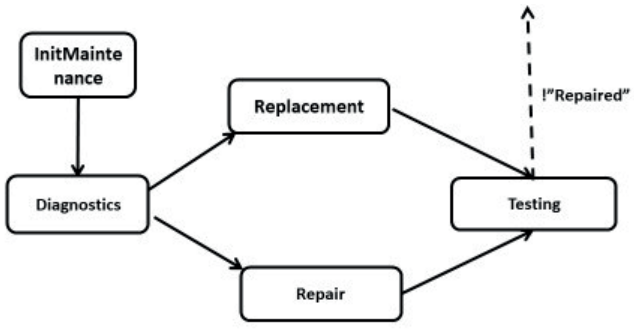
Maintenance – состояние в котором грузовик проходит техническое обслуживание

Предполагается, что в каждом из рабочих состояний грузовика проверяется его техническое состояние и в случае, когда грузовик нуждается в обслуживании *(Faulty = 1)*, он перево-



Р и с. 6. Диаграмма работы грузовика, в каждом рабочем состоянии для которого возможно возникновение неисправностей и срочных обращений в службу техподдержки Maintenance
F i g. 6. Truck operation diagram, in each operating state for which malfunctions and urgent calls to the technical support service may occur

Автомат Maintenance, получив управление и информацию о вызвавшем его состоянии грузовика, сначала выполнит детальную диагностику грузовика, чтобы решить вопрос о том, можно ли его оперативно отремонтировать, не нарушая производственного цикла. Если такой оперативный ремонт невозможен, то выполняется замена грузовика (состояние ***Replacement***). В любом случае, был ли грузовик отремонтирован или заменен, он проверяется (состояние ***Testing***). После чего автомат *Maintenance* завершает работу, что вызывает возврат управления в точку вызова с установкой признака отсутствия неисправности для продолжения производственного цикла грузовика.



Р и с. 7. Диаграмма работы автомата службы поддержки Maintenance
F i g. 7. Diagram of the support service machine operation

Таким образом, создан новый язык графического моделирования поведенческих аспектов динамических событийных систем, характеризующийся полнотой средств описания динамических аспектов сложных систем, поддерживающий структурированные технологии разработки моделей, имеющий формальную семантику моделирования в виде верифицированного алгоритма абстрактного симулятора, легко реализуемый в среде языков программирования, и поддерживающий метод расширения проблемно-ориентированными возможностями базовых сред программирования посредством их обогащения.

3. Особенности реализации и состав инструментальных средств автоматизации процесса создания цифровых моделей производства на основе языка DLAA

В процессе создания цифровых моделей на основе описанного выше графического языка описания поведенческих аспектов динамических систем DLAA используется набор специально разработанных инструментальных средств.

Системообразующую роль среди этих инструментальных средств играет штатный симулятор, реализующий операционную семантику языка DLAA, выполняя упорядоченные на модельной временной структуре вызовы функций переходов и выводов, организуя тем самым квазипараллельное исполнение автоматов-компонентов в модельном времени.

Важным инструментальным средством является графический редактор, который служит для ввода и редактирования диаграмм автоматов на языке DLAA. Этот редактор позволяет пользователю создавать и настраивать визуальные представления автоматов, что значительно облегчает процесс проектирования и анализа моделей.

Также ключевым инструментом является метатранслятор, который выполняет преобразование описание модели на метаязыке во внутреннее представление модели на базовом языке.

Таким образом, совокупность описанных инструментальных

средств составляет необходимый арсенал инструментов для автоматизации процесса создания цифровых моделей с использованием языка DLAA.

Язык DLAA и рассмотренные инструментальные средства использовались для разработки цифровых моделей для горнодобывающей отрасли.

Заключение

В статье приведён анализ известных подходов для описания поведенческих аспектов систем на основе конечных автоматов. В частности, проанализированы: классические виды конечных автоматов Мили и Мура, и их графическое представление в виде диаграмм Мура; абстрактная модель дискретной системы Уаймора; автоматы Д. Харела, представляющие собой расширение традиционного формализма конечных автоматов в виде диаграмм состояний *(Statecharts)*; автоматное программирование А.А. Шалыто – подход, ориентированный прежде всего на разработку управляющих модулей систем со сложным поведением; конечные автоматы в языках SysML и UML; *DEVS (Discrete-Event Modelling and Simulation)* – наиболее мощный формализм для моделирования сложных динамических систем с использованием абстракции дискретных событий. Анализ вышеуказанных подходов показал актуальность разработки нового графического языка спецификации реактивного поведения ЦД, предназначенного для расширения инструментальных средств реализации программного обеспечения автоматизации производства и их двойников средствами моделирования. В связи с чем выполнена поэтапная разработка нового графического языка, названного DLAA, характеризующегося компактным и, одновременно, полным набором базовых средств описания поведенческих аспектов событийных систем и их конфигураций, включением таких технологических возможностей как структурированная разработка диаграмм. Кроме этого данный язык прост в реализации и поддерживает требование реализации как инструмента обогащения базовых средств разработки производственного программного обеспечения. Выполненное расширение конструкций языка введением в них семантических функций, называемых, следуя языку SysML, активностями, поддерживает погружение языка в среду программирования базового языка разработки производственного программного обеспечения. Приведено краткое описание особенностей реализации и состава инструментальных средств автоматизации процесса создания цифровых моделей ЦД производства с использованием языка DLAA.

Список использованных источников

[1] Gapanovich D. A., Sukhomlin V. A. Tools for Constructing Production Digital Twin Models Based on an Algebraic Approach and a Graphical State Language Extended by Functional and Operational Semantics // Mathematical Modeling and Supercomputer Technologies. MMST 2024. Communications in Computer and Information Science ; ed. by D. Balandin, K. Barkalov, I. Meyerov. Vol. 2363. Cham: Springer, 2025. P. 3-16. https://doi.org/10.1007/978-3-031-80457-1_1

[2] Анализ подходов архитектурного проектирования цифровых двойников / Д. А. Гапанович, В. А. Тарасова, В. А. Сухомлин, В. П. Куприяновский // International Journal of Open Information Technologies. 2022. Т. 10, № 4. С. 71-83. EDN: ATNRSF

[3] Grieves M., Vickers J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems // Transdisciplinary Perspectives on Complex Systems ; ed. by J. Kahlen, S. Flumerfelt, A. Alves. Cham: Springer, 2017. P. 85-113. https://doi.org/10.1007/978-3-319-38756-7_4

[4] Lim K. Y. H., Zheng P., Chen C. H. A state-of-the-art survey of Digital Twin: techniques, engineering product lifecycle management and business innovation perspectives // Journal of Intelligent Manufacturing. 2020. Vol. 31, issue 6. P. 1313-1337. https://doi.org/10.1007/s10845-019-01512-w

[5] Towards the application of machine learning in digital twin technology: a multi-scale review / L. Nele [et al.] // Discover Applied Sciences. 2024. Vol. 6. Article number: 502. https://doi.org/10.1007/s42452-024-06206-4

[6] Cyber-physical integration for moving digital factories forward towards smart manufacturing: a survey / Y. Cheng [et al.] // International Journal of Advanced Manufacturing Technology. 2018. Vol. 97, issue 1-4. P. 1209-1221. https://doi.org/10.1007/s00170-018-2001-2

[7] Куприяновский В. П., Намиот Д. Е., Синягов С. А. Кибер-физические системы как основа цифровой экономики // International Journal of Open Information Technologies. 2016. Т. 4, № 2. С. 18-25. EDN: VKCXLH

[8] Review on cyber-physical systems / Y. Liu [et al.] // IEEE/CAA Journal of Automatica Sinica. 2017. Vol. 4, no. 1. P. 27-40. https://doi.org/10.1109/JAS.2017.7510349

[9] Гапанович Д. А., Сухомлин В. А. Алгебра конечных автоматов как математическая модель цифрового двойника умного производства // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 2. С. 353-366. https://doi.org/10.25559/SITITO.18.202202.353-366

[10] Гапанович Д. А., Сухомлин В. А. Моделирование функционирования шахты средствами алгебры конечных автоматов DTA // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 3. С. 634-643. https://doi.org/10.25559/SITITO.18.202203.634-643

[11] Сухомлин В. А., Намиот Д. Е., Гапанович Д. А. Анализ тенденций развития цифровых двойников нового поколения // International Journal of Open Information Technologies. 2024. Т. 12, № 7. С. 119-130. EDN: YFCLIS

[12] Глушков В. М. Абстрактная теория автоматов // Успехи математических наук. 1961. Т. 16, № 5(101). С. 3-62.

[13] Wymore A. W. Model-Based Systems Engineering. Boca Raton: FL CRC Press, Inc., 2018. 710 p. https://doi.org/10.1201/9780203746936

[14] Ören T., Zeigler B.P. System theoretic foundations of modeling and simulation: a historic perspective and the legacy of A Wayne Wymore // Simulation. 2012. Vol. 88, issue 9. P. 1033-1046. https://doi.org/10.1177/0037549712450360
[15] Harel D. Statecharts: a visual formalism for complex systems // Science of Computer Programming. 1987. Vol. 8, issue 3. P. 231-274. https://doi.org/10.1016/0167-6423(87)90035-9
[16] Канжелев С. Ю., Шалыто А. А. Автоматическая генерация автоматного кода // Информационно-управляющие системы. 2006. № 6. С. 35-42. EDN: IBLVLH
[17] Friedenthal S., Moore A., Steiner R. A Practical Guide to SysML. 3rd ed. The Systems Modeling Language. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014. 630 p.
[18] Formalizing UML State Machines for Automated Verification – A Survey / É. André, S. Liu [et al.] // ACM Computing Surveys. 2023. Vol. 55, issue 13s. Article number: 277. https://doi.org/10.1145/3579821
[19] Bouwman M., Luttik B., van der Wal D. A Formalisation of SysML State Machines in mCRL2 // Formal Techniques for Distributed Objects, Components, and Systems. FORTE 2021. Lecture Notes in Computer Science ; ed. by K. Peters K., T. A. C. Willemse. Vol. 12719. Cham: Springer, 2021. P. 42-59. https://doi.org/10.1007/978-3-030-78089-0_3
[20] Al-Fedaghi S. Modeling the Semantics of States and State Machines // Journal of Computer Science. 2020. Vol. 16, issue 7. P. 891-905. https://doi.org/10.3844/jcssp.2020.891.905
[21] Zeigler B. P., Praehofer H., Kim T. G. Theory of Modeling and Simulation. 2nd ed. Academic Press, 2000. 510 p.
[22] Wainer G. A. Discrete-Event Modeling and Simulation: A Practitioner's Approach. Boca Raton: CRC Press, 2017. 520 p. https://doi.org/10.1201/9781420053371
[23] van Tendeloo Y., Vangheluwe H. DEVS: Discrete-Event Modelling and Simulation for Performance Analysis of Resource-Constrained Systems // Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems ; ed. by P. Carreira, V. Amaral, H. Vangheluwe. Cham: Springer, 2020. P. 127-153. https://doi.org/10.1007/978-3-030-43946-0_5
[24] Zeigler B. P. Closure under coupling: concept, proofs, DEVS recent examples (wip). In: Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences (ICCES'18). Article number: 7. New York, NY, USA: Association for Computing Machinery, 2018. https://doi.org/10.1145/3213187.3213194
[25] Hong K. J., Kim T. G. DEVSpecL: DEVS specification language for modeling, simulation and analysis of discrete event systems // Information and Software Technology. 2006. Vol. 48, issue 4. P. 221-234. https://doi.org/10.1016/j.infsof.2005.04.008

Поступила 18.01.2025; одобрена после рецензирования 11.03.2025; принята к публикации 07.04.2025.

Об авторах:
Гапанович Дмитрий Антонович, ведущий программист лаборатории открытых информационных технологий кафедры информационной безопасности факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М.В. Ломоносова» (119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1), **ORCID: https://orcid.org/0000-0002-3222-694X**, dim.gapanovich@gmail.com
Сухомлин Владимир Александрович, заведующий лабораторией открытых информационных технологий кафедры информационной безопасности факультета вычислительной математики и кибернетики, ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова» (119991, Российская Федерация, г. Москва, ГСП-1, Ленинские горы, д. 1); ведущий научный сотрудник Института проблем информатики Российской академии наук, ФГУ «Федеральный исследовательский центр «Информатика и управление» Российской академии наук» (119333, Российская Федерация, г. Москва, ул. Вавилова, д. 44-22), доктор технических наук, профессор, **ORCID: https://orcid.org/0000-0001-9468-7138,** sukhomlin@mail.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

References

[1] Gapanovich D.A., Sukhomlin V.A. Tools for Constructing Production Digital Twin Models Based on an Algebraic Approach and a Graphical State Language Extended by Functional and Operational Semantics. In: Balandin D., Barkalov K., Meyerov I. (eds.) Mathematical Modeling and Supercomputer Technologies. MMST 2024. *Communications in Computer and Information Science.* Vol. 2363. Cham: Springer; 2025. p. 3-16. https://doi.org/10.1007/978-3-031-80457-1_1
[2] Gapanovich D.A., Tarasova V.A., Sukhomlin V.A., Kupriyanovsky V.P. Analysis of Approaches to the Architectural Design of Digital Twins. *International Journal of Open Information Technologies*. 2022;10(4):71-83. (In Russ., abstract in Eng.) EDN: ATNRSF
[3] Grieves M., Vickers J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In: Kahlen J., Flumerfelt S., Alves A. (eds.) Transdisciplinary Perspectives on Complex Systems. Cham: Springer; 2017. p. 85-113. https://doi.org/10.1007/978-3-319-38756-7_4
[4] Lim K.Y.H., Zheng P., Chen C.H. A state-of-the-art survey of Digital Twin: techniques, engineering product lifecycle management and business innovation perspectives. *Journal of Intelligent Manufacturing*. 2020;31(6):1313-1337. https://doi.org/10.1007/s10845-019-01512-w
[5] Nele L., Mattera G., Yap E.W. et al. Towards the application of machine learning in digital twin technology: a multi-scale review. *Discover Applied Sciences*. 2024;6:502. https://doi.org/10.1007/s42452-024-06206-4

[6] Cheng Y., Yongping Z., Ping J., Wenjun X., Zude Z., Tao F. Cyber-physical integration for moving digital factories forward towards smart manufacturing: a survey. *International Journal of Advanced Manufacturing Technology*. 2018;97(1-4):1209-1221. https://doi.org/10.1007/s00170-018-2001-2

[7] Kupriyanovsky V., Namiot D., Sinyagov S. Cyber-Physical Systems as a Base for Digital Economy. *International Journal of Open Information Technologies*. 2016;4(2):18-25. (In Russ., abstract in Eng.) EDN: VKCXLH

[8] Liu Y., Peng Y., Wang B., Yao S., Liu Z. Review on cyber-physical systems. *IEEE/CAA Journal of Automatica Sinica.* 2017;4(1):27-40. https://doi.org/10.1109/JAS.2017.7510349

[9] Gapanovich D.A., Sukhomlin V.A. Algebra of Finite Automata as a Mathematical Model of the Digital Twin of Smart Production. *Modern Information Technologies and IT-Education*. 2022;18(2):353-366. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.18.202202.353-366

[10] Gapanovich D.A., Sukhomlin V.A. Modeling the Functioning of the Mine Using the Algebra of Finite Automata DTA. *Modern Information Technologies and IT-Education*. 2022;18(3):634-643. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.18.202203.634-643

[11] Sukhomlin V.A., Namiot D.E., Gapanovich D.A. Analysis of Development Trends of New Generation Digital Twins. *International Journal of Open Information Technologies*. 2024;12(7):119-130. (In Russ., abstract in Eng.) EDN: YFCLIS

[12] Glushkov V.M. The abstract theory of automata. *Russian Mathematical Surveys.* 1961;16(5):1-53.

[13] Wymore A.W. Model-Based Systems Engineering. Boca Raton: FL CRC Press, Inc.; 2018. 710 p. https://doi.org/10.1201/9780203746936

[14] Ören T., Zeigler B.P. System theoretic foundations of modeling and simulation: a historic perspective and the legacy of A Wayne Wymore. *Simulation*. 2012;88(9):1033-1046. https://doi.org/10.1177/0037549712450360

[15] Harel D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*. 1987;8(3):231-274. https://doi.org/10.1016/0167-6423(87)90035-9

[16] Kanzhelev S.Yu., Shalyto A.A. Automatic generation of automaton code. *Information and Control Systems*. 2006;(6):35-42. (In Russ., abstract in Eng.) EDN: IBLVLH

[17] Friedenthal S., Moore A., Steiner R. *A Practical Guide to SysML.* Third Edition: The Systems Modeling Language. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2014. 630 p.

[18] André É., Liu S., Liu Y., Choppy C., Sun J., Dong J.S. Formalizing UML State Machines for Automated Verification – A Survey. *ACM Computing Surveys*. 2023;55(13s):277. https://doi.org/10.1145/3579821

[19] Bouwman M., Luttik B., van der Wal D. A Formalisation of SysML State Machines in mCRL2. In: Peters K., Willemse T.A.C. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. FORTE 2021. *Lecture Notes in Computer Science*. Vol. 12719. Cham: Springer; 2021. p. 42-59. https://doi.org/10.1007/978-3-030-78089-0_3

[20] Al-Fedaghi S. Modeling the Semantics of States and State Machines. *Journal of Computer Science*. 2020;16(7):891-905. https://doi.org/10.3844/jcssp.2020.891.905

[21] Zeigler B.P., Praehofer H., Kim T.G. Theory of Modeling and Simulation. 2nd ed. Academic Press; 2000. 510 p.

[22] Wainer G.A. Discrete-Event Modeling and Simulation: A Practitioner's Approach. Boca Raton: CRC Press; 2017. 520 p. https://doi.org/10.1201/9781420053371

[23] van Tendeloo Y., Vangheluwe H. DEVS: Discrete-Event Modelling and Simulation for Performance Analysis of Resource-Constrained Systems. In: Carreira P., Amaral V., Vangheluwe H. (eds.) Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems. Cham: Springer; 2020. p. 127-153. https://doi.org/10.1007/978-3-030-43946-0_5

[24] Zeigler B.P. Closure under coupling: concept, proofs, DEVS recent examples (wip). In: Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences (ICCES'18). Article number: 7. New York, NY, USA: Association for Computing Machinery; 2018. https://doi.org/10.1145/3213187.3213194

[25] Hong K.J., Kim T.G. DEVSpecL: DEVS specification language for modeling, simulation and analysis of discrete event systems. *Information and Software Technology*. 2006;48(4):221-234. https://doi.org/10.1016/j.infsof.2005.04.008

Submitted 18.01.2025; approved after reviewing 11.03.2025; accepted for publication 07.04.2025.

About the authors:

Dmitry A. Gapanovich, Lead Software Developer of the Open Information Technologies Lab, Department of Information Security, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation), **ORCID: https://orcid.org/0000-0002-3222-694X,** dim.gapanovich@gmail.com

Vladimir A. Sukhomlin, Head of the Open Information Technologies Lab, Department of Information Security, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University (1 Leninskie gory, Moscow 119991, GSP-1, Russian Federation); Leading Researcher of the Institute of Informatics Problems of the Russian Academy of Sciences, Dr. Sci. (Tech.), Professor, **ORCID: https://orcid.org/0000-0001-9468-7138,** sukhomlin@mail.ru

All authors have read and approved the final manuscript.