



Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Фонд содействия развитию интернет-медиа, ИТ-образования, человеческого потенциала
«Лига интернет-медиа»

В. А. Сухомлин, Д. А. Гапанович, В. М. Дубков, М. А. Князев

Моделирование дискретно-событийных систем средствами графического языка состояний и переходов конечных автоматов

Учебник

Москва, 2026



Владимир Сухомлин

**Моделирование дискретно-
событийных систем средствами
графического языка состояний
и переходов конечных автоматов**

«Автор»

2026

Сухомлин В. А.

Моделирование дискретно-событийных систем средствами
графического языка состояний и переходов конечных автоматов /
В. А. Сухомлин — «Автор», 2026

В книге рассматривается назначение и основные свойства визуального интерпретируемого языка DLAA, разработанного как средство структурированного описания конфигурации и динамики поведения сложных систем с использованием графических диаграмм, реализуемых поверх базовых языков программирования (C++, C#) и расширяющих описательные возможности этих языков в представлении поведенческих свойств сложных динамических систем. Рассмотрена вычислительная модель семантики периода выполнения языка DLAA. Приводится сравнение языка DLAA с известными графическими языками такими, как диаграммы Мура, диаграммы состояний Харела, диаграммы конечных автоматов языков SysML и UML, методология А.А. Шалыто, формализм DEVS. Описаны инструментальные средства, предназначенные для автоматизации процесса создания цифровых моделей с использованием языка DLAA.

© Сухомлин В. А., 2026

© Автор, 2026

Содержание

Введение	8
Глава 1. Анализ наиболее известных визуальных средств описания поведенческих аспектов систем на основе конечных автоматов	10
1.1. Автоматы Мили и Мура	11
1.2. Абстрактная модель дискретной системы Уаймора	13
1.3. Автоматы Харела	15
1.4. Автоматное программирование А.А. Шалыто	16
Выводы	21
2.1. Ядро языка	22
Конец ознакомительного фрагмента.	26

Владимир Сухомлин

Моделирование дискретно-событийных систем средствами графического языка состояний и переходов конечных автоматов

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный университет
имени М.В.Ломоносова»

Факультет вычислительной математики и кибернетики

Фонд содействия развитию интернет-медиа, ИТ-образования, человеческого потенциала
«Лига интернет-медиа»

В. А. Сухомлин, Д. А. Гапанович, В. М. Дубков, М. А. Князев

Моделирование дискретно-событийных систем средствами графического языка состояний и переходов конечных автоматов

Учебник для вузов

Рекомендовано ФУМО по укрупненной группе специальностей и направлений подготовки 02.00.00 Компьютерные и информационные науки в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлениям подготовки: «Фундаментальная информатика и информационные технологии», «Математическое обеспечение и администрирование информационных систем», «Математика и компьютерные науки»

Москва – 2026

УДК 519.713(075.8)

ББК 22.182.3я73

М 74

Р е ц е н з е н т ы:

М.А. Посыткин – член-корр. РАН, д.ф.-м.н., профессор, директор ФИЦ ИУ РАН;

Д.Е. Намиот – д.т.н., в.н.с. МГУ имени М.В.Ломоносова

Сухомлин В. А., Гапанович Д. А., Дубков В. М., Князев М. А.

М 74

Моделирование дискретно-событийных систем средствами графического языка состояний и переходов конечных автоматов : учебник / В.А. Сухомлин, Д.А. Гапанович, В.М. Дубков, М.А. Князев. – Москва : МГУ; Фонд «Лига интернет-медиа», 2026. – 100 с.

ISBN 978-5-521-25037-0

<https://doi.org/10.25559/978-5-521-25037-0>

В книге рассматривается назначение и основные свойства визуального интерпретируемого языка DLAA¹, разработанного как средство структурированного описания конфигурации и динамики поведения сложных систем с использованием графических диаграмм, реализуемых поверх базовых языков программирования (C#, C++) и расширяющих описательные возможности этих языков в представлении поведенческих свойств сложных динамических систем. Рассмотрена вычислительная модель семантики периода выполнения языка DLAA.

Приводится сравнение языка DLAA с известными графическими языками такими, как диаграммы Мура, диаграммы состояний Харела, диаграммы конечных автоматов языков SysML и UML, методология А.А. Шалыто, формализм DEVS.

Рассмотрены инструментальные средства, предназначенные для автоматизации процесса создания цифровых моделей с использованием языка DLAA, включая: штатный симулятор, реализующий алгоритм абстрактного симулятора для интерпретации поведения моделей в модельном времени; графический редактор, предназначенный для ввода и редактирования диаграмм автоматов на языке DLAA; метатранслятор, выполняющий преобразование текстового представления диаграмм на метаязыка во внутреннее представление модели, исполняемое симулятором. Приводятся пример применения языка DLAA.

Пособие предназначено для освоения студентами методов моделирования динамических систем дискретно-событийного класса при выполнении лабораторных, практических и проектных работ по курсу «Системная инженерия».

Ключевые слова: язык графического моделирования, DLAA, теория автоматов, дискретно-событийное моделирование, конечные автоматы, автоматное моделирование, диаграммы переходов и состояний, онтология, онтологическая схема, онтологическое моделирование, онтологическая платформа.

Авторы благодарят команду компании .Time and Space. – лично управляющего партнера Юрия Ивановича Волокитина, директора по развитию Дмитрия Юрьевича Волокитина и ведущего инженера-онтолога Дмитрия Вячеславовича Волокитина, а также всех разработчиков платформы OSA – за создание инструментальной среды, на которой основана настоящая работа.

УДК 519.713(075.8)
ББК 22.182.3я73

ISBN 978-5-521-25037-0 © Сухомлин В. А., Гапанович Д. А., Дубков В. М., Князев М. А., 2026

© Фонд «Лига интернет-медиа», 2026

Оглавление

Введение 5

Глава 1. Анализ наиболее известных визуальных средств описания поведенческих аспектов систем на основе конечных автоматов 7

1.1. Автоматы Мили и Мура 7

¹ 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>; 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

1.2. Абстрактная модель дискретной системы Уаймора	9
1.3. Автоматы Харела	10
1.4. Автоматное программирование А.А. Шалыто	10
1.5. Конечные автоматы в языках SysML и UML	11
1.6. DEVS — формализм для моделирования сложных динамических систем с использованием абстракции дискретных событий	12
Выводы	15
Глава 2. Определение языка DLAA, требований к языку, базовые операции языкового ядра	16
2.1. Ядро языка	16
2.2. Формализм спецификации конечного автомата языка DLAA	17
Глава 3. Графическая форма языка DLAA	22
Глава 4. Расширение языка DLAA функциональной семантикой	26
Глава 5. Структурирование состояний диаграмм	29
Глава 6. Операционная семантика языка DLAA. Абстрактный симулятор	32
6.1 Модель времени	32
6.2 Абстрактный симулятор языка DLAA	34
Глава 7. Погружение концептуальной модели языка DLAA в среду онтологической платформы	43
7.1. Платформа OSA	43
7.2. Онтологическая схема языка DLAA	44
Глава 8. Инструментальные средства автоматизации создания цифровых моделей на основе языка DLAA	48
8.1 Особенности реализации графического конструктора диаграмм языка DLAA	49
8.2 Конвертор онтологического образа модели в семантически эквивалентные конструкции базового языка	50
8.3 Особенности реализации симулятора	52
Список литературы	54
Приложение А	56
Синтаксис метаязыка AADL	56
Приложение Б	59
Пример использования языка для моделирования системы автоматов	59
Приложение В	75
Пример Кнута, модель функционирования лифта	75

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 производства служит концепция цифровых двойников (Digital Twin) или сокращенно 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, 2].

Процесс создания двойников разнообразных активов делает актуальной задачу разработки методов и инструментальных средств автоматизации разработки цифровых моделей активов, из которых и составляются их ЦД. При этом к таким методам и средствам во многих случаях предъявляются следующие требования:

Поддержка современной тенденции «разработки, основанной на моделях», имеющих наглядное графическое представление, облегчающее анализ и использование моделей.

Поддержка структурированного подхода и модельности в разработке ЦД и возможности комплексирования двойников активов в двойники систем.

Поддержка технологической совместимости инструментальных средств с производственным программным обеспечением (ППО), используемым при создании и использовании активов.

Именно с учетом этих аспектов коллективом автором выполнена разработка визуального графического языка диаграмм состояний и переходов конечных автоматов для спецификации конфигураций и реактивного поведения динамических систем дискретно-событийного класса, а также инструментальных средств моделирования, поддерживающих применение данного языка.

В первой главе приводится анализ наиболее известных средств описания поведенческих аспектов систем на основе конечных автоматов, включая: классические диаграммы Мура, абстрактная модель дискретной системы Уаймора, диаграммы состояний (Statecharts) Д. Харела, - автоматное программирование А.А. Шалыто, 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.

Во второй главе описаны понятия и возможности базового языка DLAA, расширяющего класс автоматов Мили и названный классом автоматов Мили*.

В третьей главе рассмотрено расширение конструкций языка введением в них семантических функций, называемых, следуя языку SysML, активностями. Введение семантических функций в язык, реализуемых средствами базового языка программирования, служит целям погружение языка DLAA в среду программирования базового языка.

В четвертой главе рассмотрена следующая очередь расширения языка введением еще двух механизмов это механизма структурирования диаграмм на основе введенной в алгебре DTA (Digital Twin Algebra) операции рекурсивного вызова автоматов [3], и механизма прерываний, предназначенного для прерывания выполняемой в рамках состояния автомата работы и перехода на альтернативную ветвь работы автомата.

В пятой главе описан набор инструментальных средств автоматизации процесса создания цифровых моделей производства на основе языка DLAA, включающий графический и оконный редакторы, метатранслятор и симулятор, апробированные на тестовых задачах.

В шестой главе описывается операционная семантика языка DLAA, модель системного времени и алгоритмы реализации абстрактного симулятора, реализующие семантику периода выполнения дискретных моделей.

В седьмой главе рассматривается разработка отображения концептуальной модели языка DLAA в среду онтологической платформы.

В восьмой главе описана модель системы инструментальных средств языка DLAA и входящие в ее состав инструментальные средства, включая: метаязык представления моделей на языке DLAA и соответствующий метатранслятор с данного языка в конструкции на базовом языке программирования (C++), графический конструктор в виде приложения на платформе OSA, конвертер онтологического образа модели в семантически эквивалентные конструкции на базовом языке программирования (C++), сборщик программной модели, абстрактный симулятор.

Приложение А. Синтаксис метаязыка AADL.

Приложение Б. Пример использования языка для моделирования системы автоматов.

Приложение В. Пример. Модель функционирования лифта.

Приложение Г. Инструкция по применению (на примере Приложения В).

Приложение Д.

Глава 1. Анализ наиболее известных визуальных средств описания 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. Рассмотрим кратко наиболее известные графические языки 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 [5, 6]. Конфигурация для обоих видов автоматов (с начальным состоянием q_0) одна и та же и имеет следующий вид:

$$M = \langle A, B, Q, \delta, \lambda, q_0 \rangle$$

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: $Q \times A \rightarrow Q$

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

q_0 – начальное состояние автомата q_0 .

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

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 интерпретируются 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 [7].

Далее будем предполагать, что конечные автоматы, с которыми мы будем иметь дело, являются полными и детерминированными. Детерминированные автоматы – это автоматы, для которых состояние $\delta(q, a)$ однозначно определяется парой $(q, a) \in Q \times A$, т.е. функция переходов $\delta(q, a) = q$ действительно является функцией.

Говорят, что автомат полный, когда функция переходов предполагается полностью определённой, т.е. значение функции $\delta(q, a)$ должно существовать для каждой пары

$$(q, a) \in Q \times A.$$

Конечные автоматы имеют наглядную форму представления в виде помеченных графов, называемых диаграммами Мура. Такими графами или диаграммами Мура, представляющими конечные автоматы, будем называть абстрактные 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 $E \rightarrow A \times B$, 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 $b = \lambda(q, a)$.

Для обеспечения лучшей наглядности графов помеченные параллельные дуги будут заменять одной дугой, помеченной списком пометок заменяемых дуг.

Рассмотрим автомат $M = \langle A, B, Q, \delta, \lambda, q_0 \rangle$, у которого:

$A = \{0, 1\}$,

$B = \{0, 1\}$,

$Q = \{q_0, q_1, q_2\}$,

$\{\delta(q_0, 0) = q_1, \delta(q_0, 1) = q_1, \delta(q_1, 0) = q_2, \delta(q_1, 1) = q_2, \delta(q_2, 0) = q_0, \delta(q_2, 1) = q_0\}$,

$\{\lambda(q_0, 0) = 0, \lambda(q_0, 1) = 1, \lambda(q_1, 0) = 0, \lambda(q_1, 1) = 1, \lambda(q_2, 0) = 1, \lambda(q_2, 1) = 0\}$.

Нетрудно видеть, что работа этого автомата состоит в том, чтобы инвертировать каждый третий символ входного потока.

На рис. 1 представлен граф этого автомата, в котором параллельные дуги заменены единственными дугами, помеченных списком меток, составленным из меток параллельных дуг.

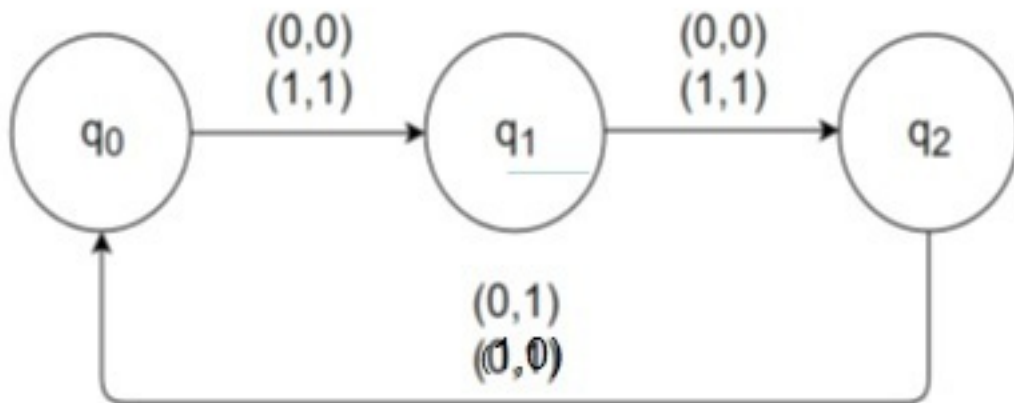


Рисунок 1. Диаграмма Мура конечного автомата, инвертирующего каждый третий символ входного потока

Теория автоматов Мили и Мура имеет важное теоретическое значение в анализе и синтезе дискретных систем. Однако для целей моделирования сложных динамических систем применяются автоматные модели с более богатыми средствами описания дискретных процессов.

1.2. Абстрактная модель дискретной системы Уаймора

К наиболее строгим с математической точки зрения подходам использования математической структуры для описания дискретных систем следует отнести исследования Уаймора [8]. 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 пятерку: $## = \{####, ####, ####, ####, ####\}$, 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 [9], исходные данные и постановка задачи примера приведены на рис. 1, где требуется построить результирующую систему $Z^* = RES(K)$ при соединении K систем Z_i для $i = \{1-4\}$ и необходимо указать множество, определяющее связь K . Пример графического изображения входных/выходных связей дискретных систем Z_1, Z_2, Z_3, Z_4 и как результат связанная система $Z^* = RES(K)$, приведены на рис. 2.

Systems	{Input ports}	Input ports which are assigned for coupling	Output functions
$K = \{$	Z1,	$\Phi, \Phi, \Phi, \Phi,$	$\Phi, \Phi, \xi_{13}, \Phi,$
	Z2,	$\Phi, \Phi, \Phi, \{P2\},$	$\Phi, \Phi, \xi_{23}, \Phi,$
	Z3,	$\{u_{32}, u_{31}\}, \Phi, \Phi,$	$\Phi, \Phi, \Phi, \xi_{34},$
	Z4,	$\Phi, \Phi, \{P4\}, \Phi,$	$\Phi, \xi_{42}, \Phi, \Phi \}$
		Z1 Z2 Z3 Z4	Z1 Z2 Z3 Z4
		Inputs come from	Output connected to

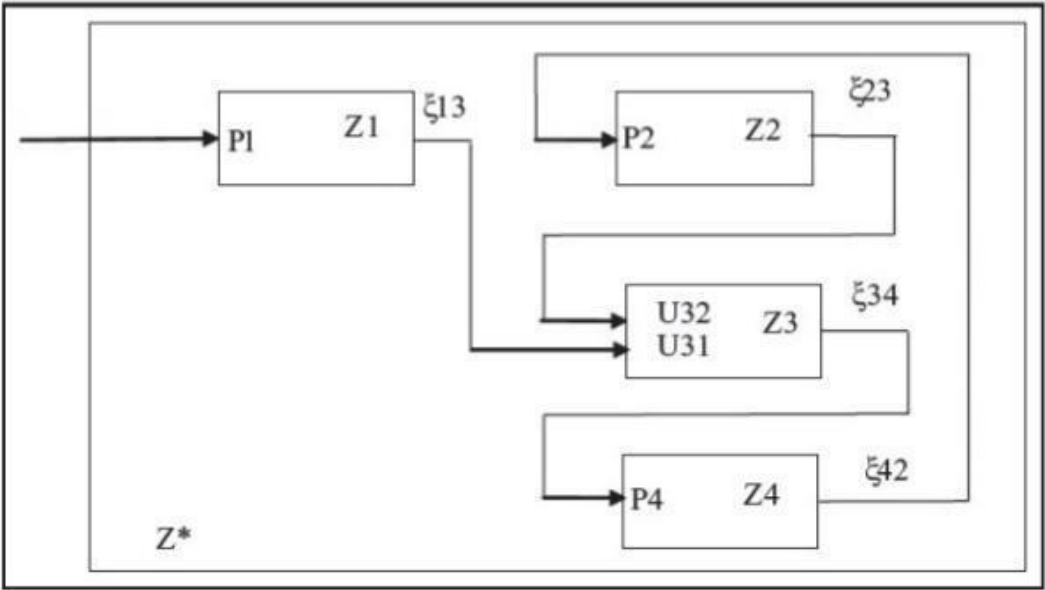


Рисунок 2. Пример связанной системы [9]

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 [10].

Диаграммы состояний Харела расширяют традиционные диаграммы Мура в следующих направлениях:

- 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 широковебательную 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 А.А. Шалыто [11]. Более точное название этого подхода «программирование с явным выделением состояний». 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 программирования 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 [12, 13]
- 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 [14, 15].

К числу таких расширений следует отнести:

- 1) семантические активности, связанные с состояниями автомата, 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) суперсостояния, представляющие собой конечные наборы состояний и вложенных суперсостояний, 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 помощью 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 [16, 17, 18, 19]. Вообще, формализм DEVS не ограничивается только моделированием дискретных систем, а охватывает и другие виды моделирования, включая: непрерывное, параллельное, сетевое и марковское (вероятностное).

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

Формализм DEVS определяет два типа моделей:

(i) атомарные модели, обеспечивающие спецификации динамики компонентов системы

(ii) 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 [18].

Атомарную модель DEVS можно рассматривать как автомат с набором состояний и функциями перехода, изменяющие состояние при возникновении внешнего события или по истечении времени. Когда никаких событий не происходит, состояние атомарной модели обновляется внутренней функцией перехода по истечении срока ее жизни. Когда происходит внешнее событие, атомарная модель изменяет свое состояние, применяя свою внешнюю функцию перехода. Время жизни состояния определяется функцией продвижения времени (time

advance function). Каждое изменение состояния может создавать выходные сообщения через функцию вывода.

Атомарные модели являются неделимыми строительными блоками модели. Основными элементами атомарной модели являются:

- 1) Набор состояний модели (S)
- 2) Продвижение времени (ta) (Time Advance)

Для каждого состояния определен тайм-аут – функция, которая должна быть определена для каждого элемента набора состояний, она детерминировано возвращает **длительность** нахождения модели в соответствующем состоянии. Длительность может быть любым неотрицательным действительным числом, включая бесконечность. В DEVS допускается длительность равная нулю, которая как правило, используется в искусственных состояниях. (Заметим, что время моделирования — это просто действительное число, и его интерпретация зависит от пользователя).

- 3) Функция внутреннего перехода (δ_{int})

$$\delta_{int}: S \rightarrow S$$

Функция внутреннего перехода δ_{int} определяет следующее состояние для каждого состояния, каждое состояние имеет не более одного следующего состояния, что предотвращает недетерминизм. Также в некоторых состояниях может не быть следующего состояния (например, если сдвиг во времени был указан как $+\infty$), а некоторые состояния идентичны следующему состоянию.

- 4) Начальное общее состояние (q_{init}).

Определяет полное начальное состояние системы, причем это не «начальное состояние (s_{init})», а именно полное состояние, которое определяет не только начальное состояние системы, но и как долго система находится в этом состоянии. Поэтому к определению исходного общего состояния добавляется прошедшее время (**Elapsed time**), чтобы обеспечить большую гибкость при моделировании системы. Симулятору будет казаться, будто модель уже некоторое время находится в исходном состоянии.

$$q_{init} : (s, e) | s \text{ из } S, 0 \leq e \leq ta(s)$$

- 5) Набор выходов (Y)

Аналогично определению набора допустимых состояний, определяется набор допустимых выходов. Этот набор служит интерфейсом для других компонентов модели системы, определяя события, которые они могут получить. События также могут обладать сложными атрибутами. Если используются порты, каждый порт имеет свой собственный набор выходов. При переходе в новое состояние используется **восклицательный знак**, обозначающий генерацию выходных данных.

$$Y: \times_{i=1}^l Y_i$$

- 6) Выходная функция (λ)

Данная функция реализует генерацию выходных событий. Функция вывода определяется по состоянию и детерминировано возвращает событие (или отсутствие события). Следует учитывать, что событие генерируется до достижения нового состояния, т.е. перед выполнением внутренней функции перехода.

Функция вывода может не возвращать никаких результатов, и в этом случае она возвращает специальный символ φ , тогда

$$\lambda: S \rightarrow Y \cup \{\varphi\}$$

- 7) Набор входов (X) – внешних событий

Определяет события, которые модель может получить извне, что также определяет интерфейс с другими моделями.

$$X = \times_{i=1}^m X_i$$

8) Внешний переход (δ_{ext})

Внешняя функция перехода зависит от текущего состояния, она также определяет новое состояние системы. Функция внешнего перехода имеет доступ еще к двум значениям; **прошедшее время** и **входное событие**. Прошедшее время указывает, сколько времени прошло для этой атомарной модели с момента последнего перехода (внутреннего или внешнего). Хотя это число было неявно известно во внутренней функции перехода (т. е. значение функции продвижения времени), здесь оно передаётся явно.

На каждом временном шаге проверяется, запланировано ли внешнее событие перед внутренним событием. Если это не так, выполняется внутренний переход. Если есть внешнее событие, которое должно произойти первым, выполняется внешний переход.

Таким образом, формальная конфигурации модели описывается с помощью восьмиэлементного кортежа следующего вида:

$$\langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

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 связанной модели используются три элемента.

1) Экземпляры модели (D)

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

$$(MS = \{M_i | i \in D\})$$

2) Спецификации модели (MS),

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

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

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. $\forall i \in D : i \notin I_i$. 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, таким образом модель, на которую оказывается влияние должна принадлежать подмножеству множества моделей в этой связанной модели. $\forall i \in D : I_i \subseteq D$.

Как следствие, связанную модель можно определить с помощью трехэлементного кортежа:

$$\langle D, MS, IS \rangle$$

Однако, чтобы связанная модель могла взаимодействовать с внешним миром, необходимо дополнить связанную модель входными и выходными событиями, которые служат интерфейсом для связанной модели. Тогда к этому кортежу добавляются компоненты X_{self} и Y_{self} ,

соответственно набор входных и выходных событий, в результате чего получается кортеж из 5 элементов:

$\langle X_{\text{self}}, Y_{\text{self}}, D, MS, IS \rangle$

Отметим еще два механизма классического DEVS:

4) Это функция разрешения конфликтов (**select**), которая используется при выполнении сложных многокомпонентных моделей, когда возникает неопределенность 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.

$ZS = \{Z_{i,j} \mid i \in D \cup \{\text{self}\}, j \in I_i\}$

Функция трансляции неявно считается тождественной функцией, если она не определена. Если событию необходимо пройти несколько соединений, все функции перевода объединяются в цепочку в порядке прохождения.

В итоге связанная модель описывается семиэлементной конфигурацией элементов следующего вида:

$\langle X_{\text{self}}, Y_{\text{self}}, D, MS, IS, ZS, \text{select} \rangle$

Выводы

Проведенный 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.

В связи с чем в последующих главах описывается новый графический язык, названный DLAA, который характеризуется компактным и, одновременно, полным набором базовых средств, включением таких технологических возможностей как структурированная разработка диаграмм, параллелизмы и аппарат прерываний. Кроме этого, данный язык прост в реализации и поддерживает требование реализации как инструмента обогащения базовых средств разработки программного обеспечения автоматизации производства.

Глава 2. Определение языка DLAA

, требований к языку, базовые операции языкового ядра

Разработка концепции и принципов построения языка во многом обусловлены требованиями, исходящими от рассматриваемых целей его применения. К основным требованиям, принятым при разработке языка DLAA, относятся следующие положения:

- язык должен быть построен по уровневому принципу, ядро языка должно содержать минимально необходимое число возможностей (операций над автоматами),
- язык должен быть достаточно универсальным, чтобы охватывать описание широкого спектра автоматных конфигураций,
- должна быть разработана формальная спецификация языка и верифицированы алгоритмы симуляции,
- язык должен включать средства, поддерживающие структурированную разработку сложных моделей,
- необходимо погружение языка и его инструментов в среду онтологической платформы, что позволит вести разработку моделей систем с применением непрерывного семантического контроля, используя логические средства платформы, обеспечивать непрерывную интеграцию создаваемых моделей с базой знаний жизненного цикла активов в масштабах всего проекта,
- семантическая часть моделей должна представлять собой совокупность именованных программных фрагментов на выбранном объектно-ориентированном языке программирования, связанных с конструктивными элементами модели и ими управляемых,
- язык и соответствующий ему инструментарий разрабатываются как средство обогащения базовых инструментальных средств программирования систем управления производственными объектами, требуется лёгкость реализации инструментальных средств языка и лёгкость их интеграции со средой программирования базового языка,
- необходимым условием является дуальность форм представления языка (как текстовой, так и графической),
- необходимость в простоте изучения и использования, так как предполагается его применение в учебных целях.

2.1. Ядро языка

Как определено выше, ядро языка должно содержать минимальное число операций композиций конечных автоматов. Отбор таких операций выполнялся на основе анализа алгебры конечных автоматов, предложенной в работе [3].

Первым кандидатом среди операций над автоматами для языка моделирования представляется операция системной композиции с согласованными событиями, т.к. она характеризуется, во-первых, значительной универсальностью, позволяя моделировать широкий класс автоматных конфигураций, и, во-вторых, является весьма экономичной при реализации (весовой характеристикой по состояниям). Более того, она допускает реализацию с линейной сложностью по числу автоматов-аргументов с помощью организации квазипараллельного способа работы входящих в нее автоматов. По существу, это универсальная операция, которой может оказаться достаточной для целей моделирования широкого класса задач.

Однако существенным недостатком операции системной композиции является общее поле событий, с помощью которых и осуществляются взаимодействие между автоматами моделируемой системы, так и переходы между состояниями внутри каждого автомата, при этом события должны быть согласованными (пользователем или инструментом), чтобы исключить сингулярность (неоднозначность), в процессе моделирования. Очевидно, что при увеличении числа автоматов в системе и числа состояний, число событий, определённых в одной области видимости, приводит систему в неуправляемое состояние.

Решением этой проблемы может быть структурирование поля событий, а именно разбиения событий на два класса: внешние для межавтоматного взаимодействия и внутренние, описывающие логику процессов работы автоматов.

Реализация этого решения приводит, во-первых, к развитию описания и реализации самой операции системной композиции, и, во-вторых, к расширению конструкции автомата Мили.

Эти вопросы решаются ниже посредством разработки формализмов определяемого языка моделирования.

2.2. Формализм спецификации конечного автомата языка

DLAA

В настоящем разделе предложенный в [3] алгебраический подход развивается в язык дискретного событийного моделирования динамических систем [19, 20]. Для адекватной поддержки задач моделирования поведения динамических систем выполняется функциональное расширение конечного автомата Мили.

Первым шагом для системы конечных автоматов введём два класса событий: внешних для межавтоматного взаимодействия и внутренних, описывающие логику процессов работы автоматов. Такое расширение автоматов Мили будет далее называться автоматами Мили* (Мили со звездочкой), при описании которых будет использоваться нотация подхода DAVS [16, 17,], как наиболее близкого к рассматриваемому случаю.

Формальное определение автомата Мили* следующее:

Определение 1: Под расширенным конечным автоматом Мили, названным Мили*, будем понимать следующую конфигурацию:

$FSM = \langle E_x, E_y, E_I, S, F, q_{init}, \delta_{int}, \delta_{ext}, \lambda_{st}, \lambda_{tr}, Ta, TS \rangle$, где (1)

E_x – множество внешних входящих событий автомата

E_y – 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 (E_x) и исходящие (E_y).

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, которое рассматривается как начало выполнения некоторого (производственного) процесса;

** - событие выхода из состояния, которое рассматривается как выход из процесса состояния.

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

δ_{int}: S → S, где S – множество состояний автомата.

Функция внутреннего перехода для каждого состояния определяет следующее состояние производственного цикла, описываемого автоматом. В некоторых состояниях следующее состояние может отсутствовать. Такие состояния считаются конечными для автомата. Они принадлежат множеству F. Также вводится возможность условных локальных переходов из одного состояния в другое. В этом случае используется решающий предикат, связанный с двумя локальными состояниями, переход в одно, из которых осуществляется в зависимости от истинности или ложности предиката. Такой механизм позволяет строить разветвлённые структуры связей между локальными состояниями автомата.

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

δ_{ext}: S × E_x → 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 [16, 17], ставящая в соответствие длительность нахождения модели в соответствующем состоянии. Длительность может быть любым действительным неотрицательным числом (в том числе случайным числом), включая бесконечность, соответствующим единицам времени структуры времени TS. Длительность равная нулю, как правило, используется в искусственных состояниях.

7) В отличие от системы DEVS 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}, если определена, может выводить вызовы исходящих сообщений в буфер вывода, когда автомат переходит в конкретное состояние. В случае, если функция для некоторого состояний не определена, считается, что функция генерирует пустое сообщение φ.

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.

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, соответствующего началу 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 TS_A . Поэтому при моделировании систем, состоящих из нескольких компонентов, представленных в модели отдельными автоматами, работающими в собственных временных структурах, важной задачей становится приведение временных структур компонентов к общей временной структуре системы. Примером здесь может служить модель центрального процессора, компоненты которого такие как, арифметическое и логическое устройство, память, кэши работают в собственных тактовых режимах.

Введённый в данном разделе класс автоматов Мили* представляет собой базовый формализм для определения языка моделирования дискретно-событийных систем, названного DLAA. В соответствии с новым понятием автомата уточним понятие системной композиции таких автоматов с согласованными внешними событиями.

Определение 2: определение операции системной композиции для расширенной версии автомата определим с помощью следующей конфигурации:

$$SM = \langle N, D, E, SPM, Q, TS, G, R \rangle, \quad (2)$$

где

N - имя модели системы SM (System_Model)

$D = \{d_i\}$ - множество ссылок/имён на автоматы (FSM), входящие в состав модели,

$i \in [1 - k]$, где k - число автоматов

E - множество всех внешних событий системы - объединение всех внешних событий (входящих и выходящих) компонентов-автоматов системы $E = \bigcup (Ex_i \cup Ey_i), i \in [1 - k]$,

$SPM = \{SP_FSM_{di}\}$ - множество спецификаций автоматов, входящих в систему, где SP_FSM_{di} - спецификация автомата $d_i \in D$,

$FSM_i = \langle Ex_i, Ey_i, E_i, S_i, F_i, q_{init}^i, \delta_{int}^i, \delta_{ext}^i, \lambda_{st}^i, \lambda_{tr}^i, Ta_i, TS \rangle$,

$Q = \{S_i\}$ - совокупность всех состояний всех автоматов системы

TS - структура времени системы,

G – блок управления системой S (используется в расширенных операциях над системами автоматов) - не обязательный элемент,

R - некоторый системный ресурс (используется в расширенных операциях над системами автоматов) - не обязательный элемент.

Рассмотренная выше операция системной композиции для расширенной версии автомата (Мили*) и будет составлять ядро языка моделирования, который в силу своего происхождения будет называться языком DLAA (Diagrams Language of Automata Algebra). Именно на основе этого ядра, состоящего из одной операции композиции автоматов, язык будет далее развиваться по уровневому принципу.

В этой операции особенно важную роль для взаимодействия автоматов системы играет аппарат внешних событий. Однако, для механизма перехода по внешнему входящему событию несет в себе семантическую неоднозначность. По существу, для целей моделирования динамических процессов нужны два различных механизма, работающих на основе внешних входящих событий.

Один механизм, типа wait/signal, применяется для синхронизации параллельных (квази-параллельных) процессов, когда, например, один автомат, находясь в некотором состоянии, ожидает сигнала от процесса, протекающего в другом автомате. В этом случае внешнее входящее событие x рассматривается именно как сигнал (возможно, несущий некоторую информацию в своих параметрах), который будет передан ожидающему состоянию данного автомата для соответствующей реакции и продолжения ожидающего этот сигнал процесса. В этом случае переход состояния в режим ожидания (wait) должен произойти до того момента времени, когда случится ожидаемое событие (signal), т.е. $t_{\text{wait}} < t_{\text{signal}}$ - условие корректного использования события x .

Второй механизм моделирует прерывание текущего процесса, протекающего в некотором состоянии автомата, с помощью внешнего события-прерывания x . Примером может служить ситуация, когда необходимо приостановить производственный процесс для проведения регламентных работ используемого оборудования (например, заправки его расходными материалами). В этом случае обработка такого события-прерывания актуальна только в течение выполнения данного производственного процесса, т.е. $[t_1, t_n)$ - интервал корректного использования события x .

Разрешение описанной выше проблемы семантической неоднозначности представленного механизма внешних входящих событий легко решается разделением обоих случаев на всех уровнях описания - онтологическом, синтаксическом и семантическом, т.е. введением отдельных понятий и примитивов для каждого из рассматриваемых случаев. Однако это ведёт к усложнению предлагаемого языка. С тем, чтобы не усложнять язык и конструкцию автоматов, в данной работе предложена реализация механизма внешних событий, поддерживающая оба рассмотренных случая.

В частности, для реализации в некотором состоянии S автомата A аналога примитивы «wait» для события x , необходимо, чтобы автомат A в функции внешнего перехода δ_{ext} содержал элемент $\delta_{\text{ext}}(S, x)=S'$, а функция длительности пребывания в состоянии S была равна бесконечности ($Ta(S)=\infty$). В этом случае обработка ситуации с внешним входящим событием x будет соответствовать семантике примитивов wait/signal.

Для случая обработки ситуации с внешним входящим событием x как событием-прерыванием достаточно в функции внешнего перехода δ_{ext} иметь реакцию на данное событие, т.е. элемент $\delta_{\text{ext}}(S, x)=S'$, и контролировать выполнение условия возникновения события x в интервале $[t_1, t_n)$.

В целом, формализм языка, описанного в данном разделе, позволяет разрабатывать модели для широкого класса систем автоматов. Для записи таких моделей разработан метаязык, описанный в Приложении А. Однако этот метаязык является языком низкого уровня, он адекватно описывает сложную конфигурацию автоматов Мили* и предназначен скорее для инструментальных средств, а не для разработчика моделей.

Именно поэтому одним из требований к разрабатываемому языку было требование дуальности языка, т.е. двух форм представления спецификаций моделей, как текстовой, так и графической.

При этом, несмотря на техническую сложность формализма языка DLAA, его графическая форма чрезвычайно проста, как для изучения языка, так и для его использования.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.