Автоматика и телемеханика, № 1, 2001

Обзоры

УДК 519.714

© 2001 г. **А. А. ШАЛЫТО, д-р техн. наук**
(Федеральный научно-производственный центр – Государственное унитарное предприятие "НПО "Аврора"", Санкт-Петербургский институт точной механики и оптики)

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ ДЛЯ СИСТЕМ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ И "РЕАКТИВНЫХ" СИСТЕМ

На основе обзора методов алгоритмизации и программирования для систем логического управления и "реактивных" систем сформулированы основные положения технологии алгоритмизации и программирования для этих классов систем, в рамках которой алгоритмы и программы рассматриваются как конечные автоматы. Дан обзор работ, посвященных применению конечных автоматов при программировании других классов задач.

1. Введение

В настоящее время в различных областях программирования все шире применяются конечные автоматы, которые в течение многих десятилетий использовались в основном при аппаратных реализациях. В настоящей работе на основе обзора методов алгоритмизации и программирования для систем логического управления и "реактивных" систем сформулированы основные положения технологии алгоритмизации и программирования для этих классов систем. При этом проектируемые программы рассматриваются как конечные автоматы. Показано, что конечные автоматы начинают применяться ведущими фирмами мира для программирования программируемых логических контроллеров (ПЛК), а также для описания поведения отдельных объектов в объектно-ориентированном программировании. Они используются при программировании протоколов, игр и схем программируемой логики, а также в такой традиционной для их применения области, как создание компиляторов.

Обзор призван показать, что существующее в настоящее время у некоторых специалистов мнение "о смерти теории автоматов" является преждевременным и ее широкое практическое применение в программировании только начинается.

2. Алгоритмизация и программирование для систем логического управления

Известен международный стандарт [1], определяющий языки программирования для ПЛК и программно реализованных ПЛК (PC-контроллеров) – промышленных (управляющих) компьютеров (обычно IBM PC совместимых) с программным обеспечением класса SoftPLC и SoftLogic для создания прикладных программ.

Известны также языки программирования для микроконтроллеров и собственно промышленных (управляющих) компьютеров [2]. Однако ведущими в области автоматизации фирмами мира [1–7], как показал обзор, выполненный в [8], до сих пор не был выбран (разработан) язык алгоритмизации для задач логического (основанного на истинности и ложности) управления, который позволил бы:

– специалистам различных областей знаний однозначно и просто понимать, что должно быть сделано, что делается и что сделано в программно реализуемом проекте;

– формально и изоморфно переходить от алгоритма к программам на различных языках программирования, используя минимальное число внутренних (управляющих) переменных в программах, так как эти переменные затрудняют понимание программ;

– легко и корректно вносить изменения в разработанные алгоритмы и в построенные по ним программы;

– корректно проводить сертификацию программ.

Отсутствие такого языка оставляло открытым вопрос о создании сквозной технологии алгоритмизации и программирования для систем логического управления, которая позволила бы повысить качество проектирования их программного обеспечения.

В [8] выполнен обзор известных технологий алгоритмизации и программирования для систем рассматриваемого класса, на основе которого была разработана новая технология, названная "SWITCH-технология", позволяющая обеспечить выполнение указанных выше требований. Эта технология может быть названа также "STATE-технология", или более точно "AUTOMATON-технология". Изложим основные положения этой технологии.

1. В качестве основного в предлагаемой технологии используется понятие "внутреннее состояние" (в дальнейшем – "состояние").

Состояния рассматриваются как некоторые абстракции, вводимые в начале процесса алгоритмизации, например путем однозначного сопоставления каждого из них с одним из физических состояний управляемого объекта, так как обычно "функционирование производственных систем проявляется через изменение их состояний [9]". При этом каждое состояние в алгоритме поддерживает объект в соответствующем состоянии, а переход в новое состояние в алгоритме приводит к переходу объекта в новое соответствующее состояние, что и обеспечивает процесс логического управления объектом.

Например, объект "клапан" может находиться в одном из четырех рабочих состояний ("закрыт", "открывается", "открыт", "закрывается"), каждое из которых может поддерживаться соответствующим состоянием в алгоритме управления. Для клапана "с памятью" алгоритм управления может иметь и три состояния, поддерживая закрытое и открытое состояния объекта одним своим состоянием (разд. 5.4.1 в [8]).

При необходимости в алгоритм управления могут быть введены также и другие состояния, связанные, например, с неисправностями объекта и неправильными действиями Оператора.

Связь состояний с внутренними (управляющими) переменными появляется в дальнейшем на этапе кодирования состояний, отсутствующем в традиционном программировании. При этом число вводимых управляющих переменных зависит от принятого вида кодирования.

Такой подход, известный из теории автоматов, принципиально отличается от подхода, обычно применяемого в программировании, при котором в ходе процесса программирования по необходимости вводятся внутренние (обычно двоичные) переменные, а затем каждый различный набор их значений объявляется состоянием программы [10]. Однако, так как понятие "состояние" в программировании прикладных задач обычно не используется, ответ на вопрос о количестве состояний в программе, содержащей, например, n двоичных внутренних переменных, остается в большинстве случаев открытым. Отметим, что в этом случае число состояний мо-

жет находиться в диапазоне от n до 2^n. При этом остается неясным также, откуда "берутся" эти переменные, сколько их должно быть и для какой цели каждая из них применяется. Отметим также, что при циклическом выполнении программы (ввиду наличия обратной связи от выхода к входу) она может быть последовательностной даже и при отсутствии управляющих переменных (гл. 13 в [8]).

2. Добавлением к понятию "состояние" понятие "входное воздействие" естественным образом вводится понятие "автомат без выхода" (автомат без выхода = состояния + входные воздействия), а после введения понятия "выходное воздействие", под которым понимается "действие" ("action") и "деятельность" ("activity"), эта формула приобретает вид: автомат = состояния + входные воздействия + выходные воздействия.

При этом соответствующая область программирования может быть названа "автоматным программированием", а процесс создания программ в этом случае – "автоматным проектированием программ".

Так же как и в [11, 12], предполагается, что действие является однократным, мгновенным и непрерываемым, а деятельность может длиться достаточно долго и возможно ее прерывание некоторым входным воздействием.

Входное воздействие в общем случае может изменить состояние; инициировать выходное воздействие без изменения состояния; инициировать выходное воздействие и изменить состояние.

3. В рассматриваемой технологии в качестве основной математической модели используется "конечный детерминированный автомат" (в дальнейшем – "автомат"); под ним понимается по классификации [13] автомат с внутренними состояниями, а не автомат с функциями поведения (Приложение 13 в [8]).

4. В качестве структурных моделей (гл. 3 в [8]) применяются автоматы без выходного преобразователя, (автоматы Мура) автоматы Мили и смешанные автоматы (С-автоматы или автоматы Мура–Мили) [14, 15].

В качестве основной структурной модели предлагается использовать автоматы Мура, в которых коды состояний и значения выходов принципиально разделены, а значения выходных переменных в каждом состоянии не зависят от входных воздействий, что упрощает внесение изменений в описания таких автоматов. Свойства алгоритмов и программ, обеспечивающие упрощение внесения корректных изменений в них, названы в [8] "управляемостью".

Первоначально число состояний в автомате Мура может быть выбрано равным числу состояний в управляемом им объекте (включая его неисправные состояния при необходимости). В дальнейшем в автомат могут вводиться дополнительные состояния, связанные, например, с неправильными действиями Оператора (Приложение 12 в [8]), а после этого число состояний в автомате может минимизироваться за счет объединения эквивалентных состояний или перехода к другой структурной модели, что, правда, без крайней необходимости делать нецелесообразно.

В автомате Мура значения выходных переменных сохраняются в течение времени его пребывания в соответствующем состоянии за счет памяти, реализующей эти состояния. В автомате Мили значения указанных переменных формируются в том числе и на соответствующем переходе, и, следовательно, для их "длительного" хранения требуется память, отличная от реализующей состояния. Поэтому в указанном смысле эти автоматы не эквивалентны.

5. Применяются автоматы первого и второго рода [8, 16]. Приоритет отдается автоматам второго рода, у которых новое состояние и значения выходных переменных формируются без задержки на "такт" (в одном программном цикле).

6. Используются различные виды кодирования состояний автоматов (разд. 3.3 в [8]): принудительное, принудительно-свободное, двоичное, двоичное логарифмиче-

ское и многозначное. Приоритет отдается многозначному кодированию состояний. Возможность применения в автоматах (ввиду наличия только одного активного состояния в каждом из них) многозначного кодирования состояний позволяет отказаться от традиционной точки зрения на автоматы как на частный случай сетей Петри, так как в последних из-за принципиальной необходимости обеспечения в общем случае одновременной активности нескольких позиций одной сети отсутствует возможность использования одной переменной для их кодирования. При этом если для автомата вне зависимости от числа состояний для их отличия достаточно только одной многозначной переменной, то, например, для частного случая сети Петри, в которой в каждой позиции не может находиться более одной метки (аналог диаграммы "Графсет" по стандарту NFC-03-190 Grafcet), число двоичных переменных, требующихся для отличия позиций, равно числу последних. В разд. 12.3 в [8] показано, что диаграмма "Графсет" с параллельными "участками" может быть заменена системой взаимосвязанных графов переходов. При этом если в первом случае число указанных двоичных переменных равно числу позиций, то во втором – число многозначных переменных равно числу графов переходов вне зависимости от числа вершин в них.

7. В качестве алгоритмической модели для автоматов с памятью при их программной реализации применяется такой непроцедурный визуальный формализм, разработанный в теории автоматов, как графы переходов, которые в литературе называются также диаграммами состояний, или диаграммами состояний и переходов ("State Diagram" или "State Transition Diagram" в англоязычной литературе).

При этом отметим, что в этих названиях, так же как и в предлагаемой технологии, отдается приоритет понятию "состояние", а не понятию "событие", являющемуся одним из наиболее распространенных в современном программировании. В излагаемом подходе событие вторично и наряду с входной переменной рассматривается в качестве разновидности входного воздействия, которое может изменить состояние.

В другом визуальном формализме (схемы (граф-схемы) алгоритмов и программ), предложенном в теоретическом программировании и являющемся процедурным, понятие "состояние" в явном виде не используется, что резко усложняет его понимаемость [17]. Это понятие не применяется и в языке регулярных выражений алгебры событий [18].

Отметим также, что графы переходов имеют плоскостное изображение, а не вертикальную направленность (как, например, схемы алгоритмов, SDL-диаграммы [8] и диаграммы "Графсет"), что существенно повышает их обозримость.

Они значительно более компактны по сравнению с эквивалентными схемами из функциональных блоков (традиционных логических элементов) и более обозримы, так как взаимодействие между графами переходов выполняется по данным, а между этими схемами – по управлению.

8. Одно из достоинств графов переходов, которые должны быть "максимально" планарными, состоит в том, что на каждой его дуге указываются не все входные воздействия (например, в форме минтермов), а только те из них, которые обеспечивают переход по этой дуге. На каждой дуге входные воздействия могут объединяться в булевы формулы, в том числе и произвольной глубины, что резко повышает компактность описания алгоритма.

Введение в модель булевых формул, как и при аппаратурной реализации (структурном синтезе последовательностных схем), расширяет классическую модель абстрактного конечного автомата, в которой дуги помечаются лишь отдельными буквами входного алфавита, и обеспечивает "параллельную" обработку входных воздействий. По сравнению с последовательной по входам программной реализацией

такое описание алгоритмов в общем случае позволяет уменьшить число состояний в реализующих их автоматах [8].

Можно утверждать, что каждое состояние в графе переходов "выделяет" из множества всех входных воздействий только то их подмножество, которое обеспечивает переходы из этого состояния, – декомпозирует указанное множество.

Изложенное позволяет применять графы переходов для решения задач логического управления при большом числе входных воздействий, значительно упрощая решение проблемы тестирования по всем путям.

9. Каждый граф переходов должен быть семантически и синтаксически корректен. Первое свойство определяет, правильная ли построена модель, а второе – правильно ли она построена [19]. При проверке синтаксической корректности граф переходов проверяется на достижимость, полноту, непротиворечивость, реализуемость и отсутствие генерирующих контуров. При проверке достижимости определяется наличие по крайней мере одного пути в графе переходов из любой его вершины в любую другую. Полнота [8] проверяется для каждой вершины графа переходов и позволяет, в частности для автоматов Мура, умалчивать пометку ее петли. При обеспечении непротиворечивости [8] каждой вершины графа переходов вместо ортогонализации могут расставляться приоритеты, что уменьшает сложность реализующей его программы. Реализуемость обеспечивается за счет различения одинаково помеченных вершин в исходном графе переходов. Генерирующие контуры [8] в графе переходов устраняются, например, за счет ортогонализации пометок дуг, образующих соответствующие контуры.

10. Дальнейшее расширение модели абстрактного конечного автомата и обеспечение "параллелизма" по выходам, как и при аппаратной реализации, достигается за счет указания в компонентах (в вершинах и/или на дугах) графа переходов не отдельных букв выходного алфавита, а значений выходных переменных (деятельностей), формируемых в этих компонентах. При этом непосредственное указание в каждой компоненте (в зависимости от выбранной структурной модели автомата) значений всех выходных переменных резко упрощает понимание алгоритма, описанного графом переходов, возможно за счет увеличения числа вершин в нем. Построенный таким образом граф переходов задает структурный конечный автомат без умолчаний значений выходных переменных. При этом отметим, что, например, для графа переходов автомата Мура при необходимости сокращения объектного кода в программе, его реализующей, неизменяющиеся при соседних переходах значения выходных переменных, указанные в вершине, в которую выполняются переходы, должны быть закомментированы.

Еще большее расширение модели конечного автомата достигается (при уменьшении ее понятности) за счет указания в компонентах графа переходов не только значений выходных переменных, но и булевых формул (возможно, автоматных, что соответствует вложенным автоматам) (разд. 12.4 в [8]).

11. "Параллелизм" по входам и выходам позволяет даже с помощью одного автомата, являющегося по определению последовательностным по состояниям (в соответствующем ему графе переходов в каждый момент времени активной является только одна вершина), реализовать параллельные процессы.

12. Поведение автоматов с памятью в отличие от автоматов без памяти зависит от предыстории, сводящейся к тому, что каждый переход в некоторое состояние зависит от одного, непосредственно связанного с ним состояния, а значения выходов, например для автоматов Мура, зависят только от состояния, в котором находится автомат.

В рамках излагаемой технологии эти свойства автоматов с памятью должны быть сохранены, а для обеспечения независимости от глубокой предыстории по состояниям и выходам графы переходов не должны содержать флагов и умолчаний соот-

ветственно [8, 17]. При этом число вершин в графе переходов должно совпадать с числом состояний в автомате. Независимость от глубокой предыстории ("будущее зависит от настоящего и не зависит от прошлого") существенно упрощает понимание [20] поведения автомата и внесение изменений как в граф переходов, так и в реализующую его программу.

Аналогичная ситуация имеет место и в марковских процессах, для исследования которых разработан весьма простой математический аппарат, который становится существенно более трудоемким для процессов, зависящих от глубокой предыстории.

13. При формальном и изоморфном переходе к программе от графа переходов без флагов и умолчаний последний является сертификационным тестом для ее проверки. При этом появляется возможность проводить сертификацию, начиная со сверки каждого графа переходов с изоморфным ему фрагментом программы. Вопросы тестирования и верификации программ, построенных с использованием конечных автоматов ("Finite State Machine" или "State Machine" в англоязычной литературе), рассмотрены в [21–28]. Кроме этих работ в пятом номере журнала "Computer" за 2000 г. описывается подход к практической верификации встроенного программного обеспечения, основанный на расширенной модели конечных автоматов, который применяется в программном продукте "VisualState". В рамках этого подхода проверяется достижимость состояний верифицируемой системы, которые, однако, задаются неявно.

14. При использовании многозначного кодирования состояний для каждого автомата с любым (конечным) их числом достаточно одной внутренней переменной, значность которой равна числу состояний.

При этом отметим, что так как (по крайней мере теоретически) любой алгоритм логического управления может быть реализован одним графом переходов, то вне зависимости от числа вершин в нем он может быть реализован программой, содержащей только одну внутреннюю (управляющую) переменную.

Отметим также, что при каждом переходе предыдущее значение многозначной переменной сбрасывается автоматически, а кроме того, ей "не с чем" состязаться.

Применять такое кодирование можно лишь в том случае, когда априори (в начале процесса алгоритмизации) известно, какое число состояний будет иметь автомат.

15. Работа с многозначными переменными поддерживается большинством языков программирования, в том числе и таким "экзотическим" языком, как язык функциональных блоков [8, 29].

Граф переходов с многозначным кодированием вершин, соответствующий выбранной структурной модели автомата, формально и изоморфно реализуется одной или двумя конструкциями switch языка СИ (гл. 5 в [8]) или их аналогами в других языках программирования, что и определило название предлагаемой технологии, а, кроме того, слово switch (переключатель) ассоциируется с теорией переключательных схем, являющейся основой теории логического управления.

Возможны два подхода к реализации графов переходов, которые могут быть обозначены следующим образом: "состояния – события" и "события – состояния".

В первом случае при реализации графа переходов используются одна или две конструкции switch по "состояниям", а во втором – каждому типу "событий" сопоставляется функция, описываемая конструкцией switch по "состояниям", отражающая переходы, инициируемые этим "событием" [30].

Если событиям соответствуют значения многозначной переменной (функции, формирующей эти значения), то в первом случае первичной является конструкция switch по "состояниям", в которую вложены конструкции switch по "событиям", а во втором – конструкция switch по "событиям", в которую вложены конструкции switch по "состояниям" (Приложение 11 в [8]). При этом отметим, что в первом слу-

чае принцип построения программ совпадает с принципом логического управления (переход из состояния в состояние под воздействием события), что резко упрощает понимание таких программ Пользователем. Отметим также, что в "развитых" науках (например, физике) понятие "пространство состояний" является одним из основополагающих [31], в то время как понятие "поток событий" таковым в этих науках не является. Например, для воды определяющими являются состояния (жидкое, твердое, газообразное), а условия переходов и тем более выполняемые ею действия вторичны.

16. Значение многозначной переменной полностью характеризует "положение" программы, реализующей один граф переходов, в пространстве ее состояний, что позволяет ввести в программирование понятие "наблюдаемость" (по одной внутренней переменной). При этом программа рассматривается как "белый ящик" с одним внутренним параметром.

Сертификация программы, построенной формально и изоморфно, например по графу переходов автомата Мура без флагов и умолчаний, может проводиться в два этапа: 1) проверка реализации всех переходов с наблюдением при этом за значениями только одной внутренней переменной; 2) проверка значений всех выходных переменных в каждом состоянии.

Графы переходов автоматов других типов всегда могут быть сведены к графам переходов эквивалентных автоматов Мура. При этом, например, каждой вершине графа переходов автомата Мили без флагов и умолчаний сопоставляются вершины графа переходов эквивалентного автомата Мура, число которых равно числу дуг, входящих в рассматриваемую вершину и помеченных различными значениями выходных переменных (разд. 4.4.3 в [8]).

Анализ всех функциональных возможностей графа переходов может выполняться с помощью графа достижимых маркировок, под которым будем понимать граф достижимых состояний (ГДС), каждой вершине которого в дальнейшем могут быть однозначно приписаны выполняемые в ней деятельности. Поэтому граф переходов автомата Мура (или автомата без выходного преобразователя) без флагов и умолчаний может использоваться в качестве ГДС, а для других типов автоматов графы переходов и ГДС различаются между собой. Так, например, для графа переходов автомата Мили без флагов и умолчаний в качестве ГДС может применяться граф переходов эквивалентного автомата Мура.

17. Из изложенного в предыдущем пункте следует, что описываемый подход позволяет снять проблему, названную в [32] расшифровкой автоматов, в [33] – распознаванием автоматов, а в [34] – идентификацией автоматов, так как считается, что автомат распознан (расшифрован, идентифицирован), если для него построен граф переходов.

В [32] показано, что автомат нераспознаваем, если он является "абсолютно черным ящиком", для которого отсутствует какая-либо информация о его внутренних состояниях. Таким образом, для автоматов с памятью тестирование типа "вход – выход", наиболее часто используемое при создании систем логического управления, проблему распознавания не решает и говорить при этом о гарантированном поведении системы управления не приходится, что, видимо, неизвестно авторам работы [35].

В [33] показано, что конечный детерминированный автомат распознаваем в результате анализа последовательности "вход – выход", если известно максимальное число состояний в минимальной (по числу состояний) форме этого автомата, а его граф переходов является сильносвязанным.

В [32] автомат, для которого известна указанная оценка числа состояний, назван "относительно черным ящиком".

18. Из изложенного в предыдущем пункте следует, что если понятие "состояние" априори не вводится, то в результате тестирования алгоритмов и программ с помощью входо-выходных последовательностей в общем случае не удается распознать автоматы, которые они реализуют.

В предлагаемой технологии состояния вводятся априори, а графы переходов строятся как сильносвязанные, что снимает проблему распознавания.

В рамках SWITCH-технологии предлагается отказаться от использования "черных ящиков" и перейти к "белым ящикам". При этом, если автомат задан в любой форме, отличной от графа переходов без флагов и умолчаний, он может быть назван "относительно белым ящиком", а если он задан графом переходов без флагов и умолчаний, то – "абсолютно белым ящиком".

Распознавание "относительно белых ящиков" может проводиться с помощью математических преобразований [8].

19. В качестве основной алгоритмической модели в рамках SWITCH-технологии предлагается применять систему взаимосвязанных графов переходов [36], что поддерживает возможность композиции и декомпозиции алгоритмов и обеспечивает практическое применение технологии при построении сложных систем логического управления.

При этом автоматы (графы переходов) могут образовывать централизованные, децентрализованные и иерархические структуры (рис. 5.41, 5.51, 5.55 в [8]).

20. Если в систему входят N графов переходов с любым числом состояний каждый, то при программной реализации даже с учетом взаимодействия графов могут использоваться только N внутренних многозначных переменных, введенных при кодировании состояний автоматов.

Для этого на программную реализацию накладывается ограничение, состоящее в том, что в каждом программном цикле в каждом графе переходов выполняется не более одного перехода: сохраняется предыдущее состояние автомата или выполняется только один переход в нем, даже в том случае, когда в рассматриваемом программном цикле в этом автомате последовательно могут быть выполнены несколько переходов.

Это обеспечивает доступность каждого значения каждой внутренней переменной для "всего окружения" – остальных $N-1$ графов системы, что позволяет не увеличивать число внутренних переменных для осуществления взаимосвязи графов переходов и делает взаимодействие графов переходов с помощью значений многозначных внутренних переменных весьма наглядным. При этом предикаты, проверяющие значения этих переменных, используются в качестве еще одной из разновидностей входных воздействий.

Таким образом, при автоматном программировании, так же как и при объектно-ориентированном программировании автоматы обмениваются сообщениями. Однако возможность взаимодействия автоматов, из которых состоит программа, за счет обмена номерами их внутренних состояний существенно отличает автоматное программирование от объектно-ориентированного, в котором объекты рассматриваются как "черные ящики", внутреннее содержание каждого из которых инкапсулировано [37].

21. Графы переходов в системе могут взаимодействовать по принципам "запрос – ответ – сброс" или "запрос – ответ", обмениваясь при этом десятичными номерами состояний.

Частным случаем такого взаимодействия является взаимосвязь головного и остальных графов переходов. При этом в случае необходимости имеется возможность запускать из головного графа параллельные процессы, реализуемые указанными графами переходов, и возвращать управление головному графу после завершения

их работы. Это расширяет область применения конечных автоматов, каждый из которых, как отмечалось выше, по определению является последовательностным по состояниям. В [8] показано, что при реализации алгоритма в виде системы взаимосвязанных графов переходов они имеют преимущества по сравнению с языком "Графсет", разработанным для описания последовательно-параллельных процессов фирмой "Телемеханик" (Франция), входящей в настоящее время в группу "Шнайдер Электрик".

22. Кроме обмена номерами состояний между автоматами, реализуемыми последовательным расположением в программе конструкций switch, графы переходов могут взаимодействовать и по принципу вложенности, что может быть реализовано вложенными конструкциями switch произвольной глубины или обращениями к функциям, построенным из этих конструкций, которые реализуют графы переходов указанной глубины вложенности.

23. Указанные принципы взаимодействия графов переходов поддерживают возможность иерархического построения алгоритмов.

Предлагаемая технология обеспечивает также проектирование алгоритмов как "сверху – вниз", так и в обратном порядке. При этом первая стратегия упрощает построение корректных алгоритмов. Так, например, если алгоритм не обладает параллелизмом по состояниям, то в первоначально построенном графе переходов каждая вершина может быть заменена фрагментом, содержащим несколько вершин; в нее могут быть вложены другие графы переходов; из нее могут вызываться другие графы переходов. В полученной структуре каждая вершина, в свою очередь, может быть детализирована. Структура программы, реализующей построенный алгоритм, должна быть изоморфна его структуре.

24. При задании алгоритма в виде системы графов переходов, последняя, если это возможно, разбивается на не связанные между собой подсистемы, образованные взаимосвязанными графами переходов, и для каждой из них, если позволяет размерность, строится граф достижимых маркировок (состояний), описывающий все функциональные возможности подсистемы.

Таким образом, в рамках предлагаемой технологии, если позволяет размерность выделяемых подсистем графов переходов, тестирование систем графов переходов и реализующих их программ может быть заменено определением всех функциональных возможностей рассматриваемого описания, как это делается при верификации протоколов [38].

При использовании сильносвязанных систем графов переходов большой размерности проектирование на основе предлагаемой технологии обеспечивает возможность построения "контролепригодных" программ, для которых может быть выполнено автоматическое протоколирование их работы в терминах автоматов [39].

25. Вводится понятие управляющего автомата, под которым понимается совокупность автомата и функциональных элементов задержки (ФЭЗ). При этом для автомата эти элементы рассматриваются наравне с объектом управления (его исполнительными механизмами и сигнализаторами): автомат кроме значений "объектных" выходных переменных формирует также значения "временных" выходных переменных, а кроме значений "объектных" входных переменных получает также значения "временных" входных переменных. "Временные" переменные отображаются на графах переходов так же, как и "объектные" переменные. В [8] рассмотрены различные подходы к программной реализации ФЭЗ, например с использованием функций (Приложение 4 в [8]).

В общем случае кроме таких функций в компонентах графов переходов могут применяться и функции других типов, например реализующие регуляторы (Приложение 10 в [8]).

26. Графы переходов могут использоваться также и при создании моделей объектов логического управления, что позволяет с единых позиций проводить описание и выполнять моделирование как разомкнутого, так и замкнутого комплекса "Управляющий алгоритм – модель объекта управления".

27. При применении излагаемой технологии спецификация на разработку управляющей программы должна состоять из замкнутой (в общем случае) схемы связей "источники информации – система взаимосвязанных автоматов – функциональные элементы задержки – исполнительные механизмы объекта управления – средства представления информации", описывающей интерфейсы автоматов, и системы взаимосвязанных графов переходов, описывающих поведение автоматов, входящих в эту схему. При этом схема связей (некоторый аналог диаграммы потоков данных в структурном системном анализе [40]) и графы переходов должны быть изображены так, чтобы их (по возможности) можно было охватить одним взглядом, что упрощает понимание спецификации (требование "гештальтности" описания [41]). Для этого, в частности, в графах переходов должны использоваться только символы переменных, а не их аббревиатуры, а смысл каждой переменной должен быть ясен при "переводе взгляда" на схему связей, которая может содержать в том числе и комментарии.

Для задач большой размерности указанная схема может строиться (с учетом всего "окружения") для каждого автомата в отдельности.

28. В рамках SWITCH-технологии предлагается иметь один язык спецификации алгоритмов (графы переходов) при применении различных языков программирования, что не было отмечено в [1].

В [8] разработаны методы формальной и изоморфной реализации графов переходов программами, написанными на различных языках программирования, которые используются в управляющих вычислительных устройствах, в том числе и в ПЛК. Для ПЛК наиболее целесообразно применять аналоги конструкции switch, позволяющие строить компактные и обозримые текстовые программы. Эту конструкцию моделирует цифровой мультиплексор, обеспечивающий возможность построения для программирования ПЛК функциональных схем, формально и изоморфно реализующих графы переходов (разд. 4.2.3 в [8]).

Графы переходов, построенные, как изложено выше, и используемые в качестве спецификации для разработки программы (определяя ее функциональные возможности), а в дальнейшем и в качестве теста для ее сертификации, могут быть реализованы также и с помощью языка "Графсет". При этом появляется возможность при сертификации программы визуально наблюдать переходы в графах переходов. Однако этот язык программирования имеет ряд недостатков. Перечислим некоторые из них: возможность одновременного отражения активности нескольких вершин в одной диаграмме приводит при реализации к двоичному их кодированию, при котором число двоичных внутренних переменных (используемых для кодирования вершин и имеющихся в ПЛК в ограниченном количестве) равно числу вершин в диаграмме; язык "Графсет" позволяет непосредственно реализовать только графы переходов автоматов Мура, требуя увеличения числа вершин для графов переходов, соответствующих другим типам автоматов; каждый граф переходов реализуется на нескольких экранах; графы переходов с переходами "назад" реализуются ненаглядно, так как для таких переходов линии, связывающие вершины, не изображаются или не могут быть изображены и поэтому заменяются "соединителями".

При этом интересно отметить, что если в ПЛК, например при реализации одной совокупности графов переходов, израсходованы все двоичные внутренние переменные, выделенные для программ на языке "Графсет", то при изменении языка программирования для реализации другой совокупности графов переходов может появиться возможность использовать другие типы внутренних переменных контрол-

лера. Таким образом, даже при программировании одного ПЛК может возникнуть необходимость в программировании на разных языках при применении одного и того же языка спецификаций.

29. Излагаемый подход позволяет Заказчику, Проектанту (Технологу), Разработчику, Программисту, Пользователю и Контролеру однозначно понимать, что должно быть сделано, что делается и что сделано в программно реализуемом проекте. Он позволяет разделять работу, а самое главное, ответственность между Специалистами различных областей знаний, а также между организациями, что особенно важно при работе с инофирмами из-за наличия языкового барьера и неоднозначности понимания естественных языков.

Подход обеспечивает возможность уже на ранних стадиях проектирования учесть все детали технического задания и продемонстрировать Заказчику в удобной для него форме, как оно понято.

30. Описываемая технология позволяет Участникам разработки проекта общаться не традиционным путем, в терминах технологического процесса (например, не "идет" режим экстренного пуска дизель-генератора), а на полностью формализованном, но понятном Специалистам различных областей знаний языке (на своего рода техническом эсперанто), на котором объясняться можно, например, следующим образом: "В третьем графе переходов, в пятой вершине, на четвертой позиции – изменить значение 0 на 1", что не вызывает разночтений, возникающих из-за неоднозначности понимания даже для одного естественного языка, а тем более для нескольких таких языков в случае, когда Участники разработки представляют разные страны, и не требует привлечения Специалистов, знающих технологический процесс, для корректного внесения изменений в программы [42].

31. Излагаемый подход позволяет снять с Программиста необходимость знания технологического процесса, а с Разработчика – тонкостей программирования.

Он дает возможность Программисту прикладных задач ничего не додумывать за Заказчика, Технолога и Разработчика, а только однозначно реализовать формализованную спецификацию, что позволяет резко снизить требования к его квалификации, а в конечном счете и вовсе отказаться от его услуг и перейти к автопрограммированию Разработчиком.

Этот подход обеспечивает возможность оставлять понятные "следы" после завершения разработки, что упрощает сопровождение и модификацию программ даже новыми Специалистами.

Он позволяет контролировать разработку и тексты программ, а не только результаты их выполнения, как это имеет место в большинстве случаев в настоящее время, приближая приемку программ к приемке аппаратуры, которая проверяется не только на функционирование [8].

32. При реализации графов переходов значения "временных" выходных переменных заменяются, например, вызовами функций, реализующих функциональные элементы задержки, а значения "объектных" выходных переменных могут заменяться вызовами других типов функций.

Поэтому рассматриваемый подход может использоваться также и при реализации управляющей части логико-вычислительных алгоритмов, как это, например, показано в Приложении 10 в [8], в котором приведена программа, обеспечивающая синхронизацию генератора с шинами главного распределительного щита.

33. Описываемая технология применяется в НПО "Аврора" при создании систем управления техническими средствами для судов различных проектов и других технологических объектов, в которых используются вычислительные средства различных фирм с разными языками программирования.

При применении одного языка спецификаций в [42] программирование выполнялось на языке ПЛ/М, в [43] – на языке функциональных блоков, в [44] – на языке ассемблера однокристальной микроЭВМ КР 1816 ВЕ51, а при создании комплексной системы управления техническими средствами для судна проекта 17310 – на языке инструкций ALPro [45, 46].

Для упрощения программирования в последнем случае Б. П. Кузнецовым (НПО "Аврора") при участии автора был создан транслятор "Ядро языка СИ – язык ALPro" [47], который применяется к программам на языке СИ, построенным формально и изоморфно по графам переходов.

34. Опыт проектирования, испытаний и эксплуатации ряда систем логического управления подтвердил эффективность SWITCH-технологии, по крайней мере, для систем рассматриваемого класса. При этом фирма "Norcontrol" (Норвегия) в [42] отметила, что применение излагаемого подхода позволило ей повысить качество системы логического управления дизель-генератором ДГР-2А 500 * 500 для судна проекта 15640.

Примеры использования этой технологии для других классов задач управления приведены в Приложениях 10 и 11 в [8].

35. Из изложенного выше следует, что наибольший эффект от применения предлагаемой технологии достигается в случае, когда алгоритмизация и программирование выполняются в разных странах [42].

36. Излагаемая технология может характеризоваться совокупностью, состоящей из семи терминов: "состояние" – "автомат" – "независимость от глубокой предыстории" – "многозначное кодирование состояний" – "система взаимосвязанных графов переходов" – "формальное и изоморфное программирование" – "конструкция switch". Ее применение обеспечивает наглядность, понятность, структурность, вызываемость, вложенность, иерархичность, управляемость и наблюдаемость программ, формально и изоморфно построенных по графам переходов.

Относительно небольшое число внутренних переменных, получаемое за счет применения многозначного кодирования состояний, резко упрощает и ускоряет их пересылку между вычислителями, которая может возникнуть, например, при построении отказоустойчивых систем, в которых при традиционном использовании, например, языка функциональных блоков приходится пересылать содержимое сотен, а то и тысяч триггеров.

37. Для задач, отличных от логического управления, следует различать состояния управляющего автомата, описываемого графом переходов, от состояний остальной памяти. Если число состояний первого типа обычно не превышает нескольких десятков, то число состояний второго типа существенно превышает эту величину [18], и поэтому они в явном виде не выделяются. Если указанное разделение состояний не выполнять, как это сделано, например, в [37], то состояния в дальнейшем и не используются, а поведение программной компоненты определяется как набор действий, выполняемых в ответ на события, без упоминания состояний первого типа, с которыми эти действия связаны. При этом следует отличать управляющие переменные от других внутренних переменных, соответствующих остальным атрибутам.

38. В рамках описываемой технологии управляющие автоматы могут строиться для:

– отдельных режимов (например автомат открытия и автомат закрытия нескольких клапанов);

– объединенных режимов (например автомат открытия и закрытия нескольких клапанов);

– индивидуальных объектов (например клапанов), реализуя для каждого из них либо отдельные, либо объединенные режимы.

Построенные автоматы могут взаимодействовать между собой непосредственно путем обмена номерами состояний, вложения и вызова, а также через дополнительные автоматы. При этом отметим, что с увеличением числа автоматов в системе доказательство правильности их совместного функционирования усложняется.

Из изложенного следует, что излагаемый подход может использоваться как при объектно-ориентированном проектировании (Приложение 8 в [8]), так и при процедурном проектировании (Приложение 5 в [8]).

39. Если исходно алгоритм реализуется граф-схемой с одним входом и одним выходом [48], то по ней с помощью методов, изложенных в [49], может быть построен граф переходов соответствующего автомата (разд. 11.5 в [8] и [50]), который в общем случае реализуется конструкцией switch, являющейся оператором в конструкции do while с условием, определяемым номером конечной вершины. Изложенный метод является более эффективным по сравнению с методом структурирования программ Ашкрофта и Манны [8]. При этом отметим, что в [51] в качестве примера приведен граф переходов, реализующий один из алгоритмов сортировки чисел, который традиционно описывается циклической граф-схемой.

40. При непроцедурном задании автоматов в виде таблиц переходов и выходов их строкам и столбцам (или наоборот) соответствуют состояния и события, и поэтому при непроцедурной реализации автоматов интерпретатором порядок обработки (состояние – событие или событие – состояние) не имеет значения (Приложение 3 в [8]). При процедурном задании автоматов в виде схем алгоритмов (программ) [52] выполняется процедурная реализация, которая обычно более экономична по объему памяти и для которой порядок обработки состояний и событий существен.

Будем называть эти схемы, начинающиеся с дешифратора событий (значений входных переменных), событийными, а схемы, начинающиеся с дешифратора состояний, – автоматными. При этом подходы к построению таких схем будем называть событийным и автоматным соответственно.

Автоматный подход позволяет отказаться от применения указанных схем, так как программирование в этом случае может выполняться непосредственно по наиболее "обозримой" форме задания автоматов – графам переходов (с многозначным кодированием вершин), которые изоморфны таким схемам с дешифратором состояний (автоматные схемы алгоритмов) и конструкции switch языка СИ (разд. 13.4 в [8], [17]).

В [8] предложены схемы алгоритмов, построенные с помощью событийного и автоматного подходов, которые реализуют R-триггер и счетный триггер соответственно. По мнению автора, автоматные схемы алгоритмов являются более "понятными".

В [53, 54] описаны программы, реализующие некоторые функции управления элементом графического пользовательского интерфейса "тулбар" и построенные на основе событийного и автоматного подходов. В первом случае, так же как и в объектном программировании, в котором "делается упор на создание автономных агентов, взаимодействующих между собой для достижения желаемого результата" [37], для каждого события реализован свой обработчик. При этом так же, как и в объектном программировании для обеспечения взаимодействия методов класса, эвристически вводятся флаговые (управляющие) переменные (в данном случае одна), оставляя вопрос о понятности и корректности последовательностной логики такой программы открытым. Во втором случае обработчики событий вызывают функцию, формально и изоморфно реализующую граф переходов, передавая ей в качестве параметра номер произошедшего события. Благодаря тому, что логика, ранее рассредоточенная по обработчикам, теперь сконцентрирована внутри одной функции, резко упрощается понимание поведения программы [54, 55].

41. Описываемая технология является сквозной и прозрачной, так как она охватывает этапы алгоритмизации и программирования, для которых разработаны мето-

ды, обеспечивающие высокое качество проектирования не только в пределах одного этапа, но и между этапами.

42. Изложенный подход подробно описан в [8], а также в [17, 56–59]. Подходы, наиболее близкие к предлагаемому, содержатся в [36, 60–62].

Разработанный подход существенно дополняет международный стандарт IEC 1131-3 (IEC 61131-3) [1], который, описывая типовые языки программирования для ПЛК, не содержит (в отличие от [8]) методов проектирования программ на этих языках. Эта проблема не решена и в документации ведущих в области автоматизации фирм мира [63], в которой в лучшем случае содержатся лишь отдельные примеры (например в [64]). Такая же ситуация имеет место и в [65–74].

Рассмотренный подход был предложен и внедрен автором в 1991 г. [43, 75]. Он был, в частности, использован в НПО "Аврора" при создании:

– системы управления дизель-генератором ДГР-2А 500 * 500 для трех судов проекта 15640 на базе аппаратуры "Selma-2" фирмы "ABB Stromberg" (Финляндия). Программирование выполнялось на языке функциональных блоков [43];

– системы управления дизель-генератором того же типа для судна проекта 15967 на базе аппаратуры "Selma-2" фирмы "ABB Stromberg". Программирование выполнялось на языке функциональных блоков;

– системы управления дизель-генератором того же типа для судна проекта 15760 на базе аппаратуры фирмы "Norcontrol" (Норвегия). Программирование выполнялось фирмой на языке ПЛ/М [42];

– комплексной системы управления техническими средствами пяти судов проекта 17310 на ПЛК "Autolog" фирмы "FF-Automation OY" (Финляндия) [45]. Программирование для общесудовых систем выполнялось на языке инструкций ALPro [46], а для системы управления вспомогательными механизмами – с помощью транслятора "Ядро языка СИ – язык инструкций ALPro" [47];

– комплекта "Авролог" для управления техническими средствами судна на ПЛК "Autolog" фирмы "FF-Automation OY". Программирование выполнялось с помощью указанного выше транслятора;

– автоматизированной системы управления технологическими процессами (АСУ ТП) центральной подготовительной станции первичной обработки нефти "Авролог-НП1" (Северо-Ореховское месторождение). Программирование выполнялось на языке инструкций ALPro;

– АСУ ТП дожимной насосной станции "Авролог-НП2" (Северо-Покурское месторождение). Программирование выполнялось на языке инструкций ALPro;

– системы управления турбокомпрессорным агрегатом "Ларина", используемой при производстве полиэтилена в ПО "Полимер" (Новополоцк). Программирование выполнялось на языке ассемблера однокристальной микро-ЭВМ КР 1816 ВЕ51 [44].

3. Применение конечных автоматов при программировании программируемых логических контроллеров

В 1993 г. фирма "Модикон" (США), входящая в настоящее время в группу "Шнайдер Электрик" [4], а в 1996 г. и фирма "Сименс" (Германия) предложили использовать графы переходов в качестве одного из языков программирования для своих ПЛК, причем последняя в [2] утверждает, что описание на таком языке не только подходит для Программиста ПЛК, но также понятно Инженеру-механику, Инженеру по запуску оборудования и Инженеру по обслуживанию. При этом отметим, что этот язык был введен указанными фирмами в практику проектирования, несмотря на то что он не входит в состав языков, рекомендуемых международным стандартом [1].

Подходы, предложенные этими фирмами, обладают по сравнению с излагаемой технологией тем преимуществом, что они разработали трансляторы непосредственно с этого языка, получив тем самым исполняемый язык спецификаций [76], а недостатки этих подходов состоят в следующем:

– эти трансляторы, естественно, не взаимозаменяемы;

– трансляторы ограничивают возможность применения моделей и подходов, отличных от принятых при их разработке;

– в документации не указаны особенности построения графов переходов, перечисленные выше, которые обеспечивают их понятность и возможность построения качественных программ на их основе;

– отсутствует указание на необходимость создания описанной выше схемы связей;

– не удается охватить даже один граф переходов "одним взглядом", так как собственно граф, условия переходов и выходные переменные изображаются на разных экранах ("негештальтное" описание);

– они используют графы переходов в качестве одного из языков программирования, не указывая при этом, что графы переходов целесообразно применять и в качестве языка алгоритмизации при использовании других языков программирования.

В настоящее время подход, основанный на применении графов переходов, названный "State Logic", начал использоваться также и фирмой "GE Fanuc Automation" (GE – General Electric) для программирования старших моделей своих ПЛК [77]. При этом был разработан язык программирования ECLiPS (English Control Language Program Software), позволяющий описывать граф переходов. При этом в материалах фирмы отмечено, что предложенный подход может применяться специалистами, которые не имеют опыта программирования. Этот подход рассматривается фирмой как практическая альтернатива лестничным схемам.

Недостатки указанного подхода состоят в следующем: графы переходов представляются как картинки, помеченные английскими словами; применяется специализированный язык программирования, использующий английский язык, а не математическую нотацию, позволяющую исключить умолчания; он требует применения специализированного процессора (State Logic Processor); не предложено использовать графы переходов в качестве языка спецификаций при программировании на языке лестничных схем младших моделей ПЛК.

Диаграммы состояний применяются в качестве одного из языков программирования промышленных контроллеров фирмы "Matsushita Automation Controls" [78], а язык "список состояний" – для программирования контроллеров фирмы "Festo Cybernetic" [79].

4. Использование конечных автоматов в программировании

До последнего времени графы переходов применялись лишь в отдельных задачах теоретического программирования [80], а в практическом программировании они в основном использовались лишь для разработки компиляторов [81, 82], в то время как для аппаратурных реализаций графов переходов давно и широко применяются [16]. В [83] было предложено использовать графы переходов в программировании, ввиду того что "мы можем рассматривать любую программу так, будто бы она полностью реализована аппаратурными средствами".

Однако при создании программ для решения функциональных (прикладных) задач до последнего десятилетия главенствующим был процедурный подход, использующий не графы переходов, а такой визуальный формализм, как схемы алгоритмов.

Ограниченность (по мнению Г. Буча [84]) этого подхода при создании сложных программ привела к разработке и широкому распространению объектно-ориенти-

рованного подхода. Однако первое время новый подход поддерживали лишь языки программирования [85], а метод проектирования программ в рамках этой парадигмы отсутствовал.

Такой метод был разработан Г. Бучем в 1991 г. [84]. Он включает несколько визуальных формализмов (диаграмм) для отображения различных особенностей выделяемых классов и объектов. При этом в [84] было предложено (на одной странице) для описания поведения объектов использовать диаграммы переходов и был приведен пример применения этих диаграмм. Однако в рамках метода Буча в отличие от излагаемого подхода эти диаграммы являлись лишь "картинками" [12], помеченными словами, а не были математическими моделями. Кроме того, Г. Бучем (в отличие от [8, 17]) не было отмечено, что при определенном подходе к построению схем алгоритмов они могут быть изоморфны графам переходов, и поэтому их противопоставление в этом случае некорректно. Этот факт не был отмечен также и в [50, 86].

Примерно та же картина, что и в книге Г. Буча, сохранялась в [87], авторы которой предложили моделировать "мир" в состояниях. При этом отметим, что этот "мир" в основном состоит из микроволновых печей, клапанов и насосов, относящихся к объектам, управляемым системами логического типа.

Существенно большее внимание диаграммам состояний и переходов, не изменив определения состояний через значения внутренних переменных, Г. Буч уделил в книге [11], разд. 5.3 которой написан на основе работ Д. Харела [88, 89], предложившего, по мнению Г. Буча, "простой, но очень выразительный подход, который гораздо эффективнее традиционных автоматов с конечным числом состояний".

Однако указанное противопоставление некорректно, так как подход Д. Харела, как и подход, излагаемый в настоящей работе, базируется на таких автоматах.

Д. Харел предложил свой подход для несколько более широкого класса систем управления по сравнению с системами логического управления, которые называются "реактивными" или "реагирующими" ("reactive") [90], "поведение которых лучше всего характеризуется их реакцией на события, произошедшие вне их контекста" [91]. Такие системы реагируют на поток событий изменением состояний и выполнением действий при переходах из состояния в состояние или действий и деятельностей в состояниях [92–100]. При этом Д. Харелом был предложен визуальный формализм (модификация диаграммы состояний и переходов), названный им "картой состояний" ("State-chart" в [88], или "State chart" в [92]), которая "математически эквивалентна диаграммам автоматов Мура–Мили" [90], но позволяет в ряде случаев описывать поведение этих автоматов более компактно. Карты состояний могут быть также названы как "диаграммы Харела". Отметим основные особенности этих диаграмм.

Наряду с состояниями могут использоваться "гиперсостояния (суперсостояния [12]), объединяющие несколько состояний, имеющих идентичную реакцию на одно и то же событие" [101]. При этом вместо изображения таких переходов в некоторое состояние из всех состояний, охватываемых гиперсостоянием, изображается только один переход из гиперсостояния в указанное состояние (обобщение переходов [101]).

Гиперсостояния теоретически могут иметь произвольную глубину вложения. Переходы из гиперсостояния связаны со всеми уровнями вложенности в него.

Гиперсостояния могут объединять ИЛИ-состояния (последовательные состояния) и И-состояния (параллельные состояния) [91]. В первом случае, перейдя в гиперсостояние, автомат может находиться только в одном из состояний (или в одном состоянии или в другом), а во втором случае в гиперсостоянии "возникает необходимость в параллельных состояниях" [91], изображаемых в "ортогональных регионах".

В вершинах диаграмм, соответствующих состояниям или гиперсостояниям, могут указываться: деятельности (помечаются словом do); действия, выполняемые однократно при входе в вершину (помечаются словом entry); действия, выполняемые однократно при выходе из вершины (помечаются словом exit).

Два последних случая могут быть названы обобщением действий, так как при этом действия, однократно выполняемые при входе в вершину, заменяют одинаковые действия, которые в автоматах Мили или смешанных автоматах указываются на всех входящих в нее дугах, соответствующих переходам, а действия, однократно выполняемые при выходе из вершины, заменяют одинаковые действия, которые в автоматах Мили или смешанных автоматах указываются на всех выходящих из нее дугах, соответствующих переходам.

На каждой дуге диаграммы может быть указано событие, вызывающее переход, и логическое условие, "охраняющее" этот переход. На дуге также могут быть указаны действия, однократно выполняемые на переходе, и формируемые на переходе события.

В диаграммах могут применяться также псевдосостояния, например условное (C), терминальное (T) и историческое (H), которые не являются реальными состояниями.

Если гиперсостояние содержит историческое псевдосостояние, то при переходе в это гиперсостояние "управление передается тому состоянию, в котором система находилась в данном гиперсостоянии в последний раз" [101]. Использование обобщенных переходов и исторического псевдосостояния позволяет, например, эффективно специфицировать прерывание, связанное с переходом системы из любого состояния, принадлежащего гиперсостоянию, в некоторое другое состояние, в котором выполняется обработка прерывания, и продолжение работы системы (после обработки прерывания) из того состояния, в котором наступило прерывание [101].

Некоторые из предложений Д. Харела, например обобщение переходов, могут применяться и в рамках SWITCH-технологии.

Несмотря на некоторую похожесть подхода Д. Харела и излагаемого, они имеют принципиальные отличия, состоящие в том, что, во-первых, у Д. Харела могут использоваться как состояния, так и гиперсостояния, в то время как в SWITCH-технологии применяются только состояния, а во-вторых, при наличии гиперсостояний, по крайней мере с ИЛИ-состояниями, считается, что такая диаграмма Харела описывает поведение одного автомата, в то время как в SWITCH-технологии в этом случае применяется система взаимосвязанных графов переходов, описывающих функционирование системы взаимосвязанных автоматов.

При этом, во-первых, не ясно, как с помощью нотации Харела изобразить диаграмму при большом числе гиперсостояний с большим уровнем вложенности, а во-вторых, видимо по этой причине в объектном моделировании считается, что диаграмма Харела (диаграмма состояний) описывает жизненный цикл отдельного объекта, а для моделирования поведения сообщества совместно работающих объектов предлагается использовать диаграммы другого типа (диаграммы взаимодействий) [91], что резко усложняет формальный переход к реализации.

В излагаемой технологии вместо понятия "объект" применяются понятие "автомат" и динамические диаграммы одного типа (система взаимосвязанных графов переходов) даже при наличии большого числа автоматов с большим уровнем вложенности, которые весьма просто формально и изоморфно реализуются практически на любом языке программирования.

Кроме того, при объектном подходе остается неясной формальная связь между статическими и динамическими диаграммами [90, 91], в то время как при замене диаграммы объектов схемой взаимодействия автоматов и использовании схемы связей, включающей систему взаимосвязанных автоматов, и системы взаимосвязанных

графов переходов вся концепция в целом оказывается выраженной в терминах автоматов.

Видимо по этим причинам, а также из-за сложности и специфичности нотации диаграммы Харела в отличие от графов переходов не применяются до настоящего времени при программировании ПЛК.

SWITCH-технология может рассматриваться в качестве методологии проектирования программного обеспечения, по крайней мере, для класса SoftLogic [72]. Этот подход позволяет "не говорить о том, что программное обеспечение работает, а объяснить, почему оно работает" [20]. Подход, в частности, позволяет ответить на три вопроса: откуда появляются внутренние (управляющие) переменные?, сколько их должно быть?" и для какой цели каждая из них используется?

Автоматный подход применяется для спецификации протоколов сетей ЭВМ [38, 102–105], а также в SDL-методологии [106–108], разработанной Международной комиссией по телефонии и телеграфии для создания программного обеспечения телекоммуникационных систем. Однако SDL-диаграммы, являющиеся некоторой разновидностью схем алгоритмов, в которые в явном виде могут вводиться состояния, весьма громоздки и соответствуют только одному классу автоматов – автоматам Мили. В SDL-методологии не оговаривается ряд вопросов, отмеченных выше, которые связаны, например, с умолчаниями и флагами. Ряд других недостатков SDL-диаграмм (в частности их вертикальная направленность, существенно уменьшающая обозримость спецификации и не позволяющая эффективно использовать плоскость листа бумаги или дисплея) отмечен в [109].

Это привело к тому, что при разработке [109–112] технологии программирования встроенных систем реального времени, реализующих широкий класс алгоритмов, ее авторы, отметив недостатки модели Харела, например, "отсутствие ориентации на генерацию конечного кода", при создании модели поведения объектов объединили достоинства SDL-диаграмм и диаграмм Харела, сохранив, правда, альтернативные нотации каждого из них. Та же ситуация сохранилась и при создании объектно-ориентированной методологии разработки программного обеспечения систем реального времени [113]: в поведенческой модели используются диаграммы состояний в модифицированной нотации Харела, а для детализации – SDL-диаграммы [114]. Это делает модели, предложенные в [109, 114], весьма специфичными и ограничивает их применение для задач логического управления технологическими процессами, при решении которых SDL-диаграммы (в отличие от телефонии) ведущими в этой области фирмами мира не используются.

Еще более разнообразной является модель поведения, предложенная при разработке унифицированного языка моделирования (Unified Modeling Language (UML)) [12, 91, 92, 115, 116], являющегося, по мнению его авторов, коллекцией наилучших инженерных подходов, применяемых для объектного моделирования сложных систем общего назначения. Построение этого языка базируется на подходах, предложенных в [11, 84, 117, 118].

В этом языке для описания поведения объектов используется "конечный автомат" ("State Machine"), который в зависимости от приоритета состояний или деятельностей в этом описании задается диаграммой состояний (State Diagram) или диаграммой деятельностей (Activity Diagram) соответственно, что некорректно, так как в последних возможен параллелизм по переходам в одной диаграмме. При этом диаграмма состояний не является одной из классических, а считается эквивалентной диаграмме Харела [90], но почему-то несколько отличается от нее, [92].

Наличие в языке диаграмм деятельностей, "являющихся одной из самых больших неожиданностей UML" [12], которые, как отмечено выше, в рамках теории автоматов не могут рассматриваться как конечные автоматы, создает неопределенность при выборе нотации для описания поведения объектов. При этом отметим,

что эти диаграммы сильно напоминают диаграммы "Графсет", которые уже более двух десятилетий используются при программировании систем автоматизации, но отличаются (в худшую сторону) от последних отсутствием "полочек" для указания в явном виде условий, при которых выполняются переходы, и наличием условных вершин, характерных для схем алгоритмов. Отметим также, что если в диаграммах "Графсет" вершины названы этапами, то в диаграммах деятельностей аналогичные вершины некорректно названы состояниями.

Кроме того, в [15], в отличие от [12, 92], показано, что каждая диаграмма деятельностей (так же как и каждая диаграмма "Графсет" (п. 21)) может быть заменена системой взаимосвязанных графов переходов, которые, как отмечено в п. 6 для диаграмм "Графсет", могут быть реализованы с меньшим числом внутренних переменных.

Неопределенность с выбором нотации для описания поведения объектов в UML еще более увеличивается, в связи с тем что к диаграммам деятельностей отнесены также и конструкции, аналогичные SDL-диаграммам, что свидетельствует о недостаточном научном обосновании формирования рассмотренной части "коллекции".

Для большего однообразия фирма "I-Logix", принимавшая участие в разработке указанного языка, при создании методологии проектирования объектно-ориентированных встроенных систем реального времени выбрала из этой части "коллекции" только диаграммы Харела в их первозданной нотации [99], но, однако, не решила и даже не поставила вопрос о сертификации поведении совокупности таких диаграмм.

Отметим также, что диаграммы Харела используются в [119] и при визуальной разработке программных компонент, описываемых конечными автоматами и предназначенных для различных целевых платформ. Только эти диаграммы применяются и для описания поведения объектов в [181].

Расширение нотации диаграмм Харела выполнено в [120]. Если в диаграммах Харела на переходах допустимо применение лишь отдельных условных вершин, характерных для схем алгоритмов (Flow Diagram), в которых понятие "состояние" не используется (stateless), то в [120] эти вершины могут образовывать более сложные конфигурации. Получающаяся при этом тесная связь между "state diagram" и "flow diagram" привела к появлению нового термина "stateflow". Возможность реализации этих диаграмм в качестве исполняемых спецификаций на широко используемом (по крайней мере, в университетских кругах) языке "MATLAB" [121] позволяет также реализовывать эти диаграммы на языках C и C++.

Однако, так как и эта нотация является весьма специфической, то для систем логического управления, принадлежащих к нижнему уровню управляющих систем (особенно при реализации их на ПЛК), более целесообразно при алгоритмизации применять системы взаимосвязанных графов переходов, содержащих минимальную номенклатуру компонент и обозначений ("не надо размножать сущности без необходимости" (Оккам)) и ориентированных на классические модели теории автоматов, по которым, как отмечено выше, программирование может выполняться формально и изоморфно даже вручную.

Для этого класса систем предлагаемая технология позволяет строить математические модели, по которым, в частности, при любом типе двоичного кодирования состояний для программной реализации могут быть построены системы булевых формул, в то время как при других подходах такая задача даже не ставится.

Предлагаемая технология поддерживает проектирование и реализацию программного обеспечения для рассматриваемого класса систем и определяет состав и содержание выпускаемой документации, которая должна контролироваться Заказчиком.

Изложенное весьма актуально, так как эта документация в лучшем случае содержит только тексты программ [122], их описания и руководство пользователя, а в худшем – тексты программ в документацию не включаются [123].

Кроме перечисленных выше работ конечные автоматы, рассматриваемые в [182] как разновидность процессов, в настоящее время применяются при управлении автоматическими выключателями корабельных электроэнергетических систем [124], при создании непроцедурных языков программирования для автоматизированных систем управления технологическими процессами [125], при поиске подстрок [126] и для описания поведения "объектов" [15, 127] при программной реализации сложных систем [87, 128–147], в том числе "многониточных" контроллеров [30], а также при решении других задач, связанных с параллельными процессами (например, задачи о пяти философах, собравшихся за круглым столом и имеющих "проблемы" с вилками [148], о функционировании мячиков внутри окна [149], о синхронизации цепи стрелков [150]).

Графы переходов являются также основной формой описания управляющих последовательностных процессов при создании программного обеспечения систем обработки информации на основе методов структурного системного анализа и проектирования [40]. Они используются и для документирования, проводимого по методологии IDEF (ICAM (Интеграция Компьютерных и Промышленных Технологий) DEFinition (Определение)), трансформаций объектов, происходящих в ходе выполнения технологического процесса [74].

Роль и место состояний ("состояния бытия"), конечных автоматов, графов переходов и конструкции switch при программировании игр описаны в [151].

Более того, графы переходов используются для описания поведения машин Тьюринга, применяемых при формальном определении понятия алгоритм [101, 152]. Поэтому заданная машина Тьюринга также может быть реализована на основе конструкции switch, в которой используются дополнительные действия, имитирующие сдвиг головки "чтение–запись" по рабочей ленте. Таким образом, излагаемый подход в принципе может применяться и для более широкого по сравнению с рассматриваемым класса задач, как например это делается при использовании модели "расширенного конечного автомата", который рассматривается в виде композиции двух автоматов – конечного и контекстного, состояния первого из которых называются основными и кодируются одной переменной, а состояния второго отражают переменные, называемые контекстными [38, 102]. Этот же подход может применяться при описании "объектов" в объектно-ориентированном программировании, что позволяет в явном виде поддержать одно из определений объекта как некоторого элемента, на который можно воздействовать, изменяя его состояния.

Из изложенного следует, что в настоящее время автоматный подход еще только начинает использоваться при программной реализации алгоритмов, и утверждение, высказанное в [153], о "смерти" теории автоматов является, по мнению автора, сильно преувеличенным. Более того, рецензент книги [8] определяет в [123] "конечные автоматы как инструмент борьбы с монополизмом фирм, разрабатывающих программное обеспечение", а в [50, 86] и вовсе приведен "гимн" применению автоматов в программировании.

В этой связи особо следует отметить, что в [154] создатель операционной системы Unix К. Томпсон на вопрос о текущей работе ответил: "Мы создали язык генерации машин с конечным числом состояний, так как реальный селекторный телефонный разговор – это группа взаимодействующих машин с конечным числом состояний". Этот язык применяется в "Bell Labs" по прямому назначению – для создания указанных машин, а кроме того, с его помощью стали разрабатывать драйверы.

5. Алгоритмизация и программирование "реактивных" систем

Технология, описанная в разд. 2, предназначена для создания алгоритмического и программного обеспечения систем логического управления, в которых ввод входных переменных выполняется только путем опроса [155, 156] – используется "цикли-

ческий исполнитель" [91]. Другая особенность этой технологии состоит в том, что при ее применении программы реализуются компилятивно: по графам переходов строятся конструкции switch или их аналоги, которые выполняются непосредственно и поэтому являются "активными".

А. А. Грабовский (АО "Каскод", Санкт-Петербург) применил автоматный подход, изложенный в [8], для программной реализации логической части систем управления, проектируемых на микроконтроллерах фирмы "Сименс", для которых характерен развитый механизм обработки прерываний. При этом по таблично заданным графам переходов автоматически строится массив (напоминающий конструкции switch), который является "пассивным" и обрабатывается интерпретатором, учитывающим возможность появления запросов на обработку во время, когда микроконтроллер реализует соответствующий переход в одном из графов.

Этот подход использовал также и С. Б. Терентьев (НПО "Аврора") при реализации протоколов в распределенных управляющих системах, построенных на основе указанных микроконтроллеров. При этом написание программ выполнялось формально и изоморфно непосредственно по построенным графам переходов.

Н. И. Туккель (НПО "Аврора") и автор применили SWITCH-технологию при разработке системы управления дизель-генератором, реализуемой на промышленном компьютере и операционной системе QNX, в которой управляющая программа выполняется как один процесс, а программа, моделирующая объект управления, – как другой процесс. При этом был создан вариант SWITCH-технологии для "реактивных" систем [39] ввиду того, что управление на основе событий приводит к существенным отличиям при проектировании системы по сравнению со случаем применения в ней только "циклического исполнителя" [91]. Такие системы обычно реализуются на промышленных компьютерах, работающих под управлением операционных систем реального времени.

Этот вариант технологии характеризуется следующими особенностями:

– в качестве базового используется понятие "автомат", а не "класс", "объект", "алгоритм" или "агент", как это имеет место при других подходах. При этом соответствующая область программирования может быть названа "автоматно-ориентированное программирование";

– в общем случае автоматы рассматриваются не изолированно, а как составные части взаимосвязанной системы – системы взаимосвязанных автоматов, поведение которой формализуется с помощью системы взаимосвязанных графов переходов;

– в качестве основной применяется модель смешанного автомата, для описания поведения которого используется соответствующий граф переходов только с "простыми" состояниями (гиперсостояния не используются);

– расширена (по сравнению с [8]) нотация, применяемая при построении графов переходов (например в части перечисления вложенных автоматов);

– на этапе изучения предметной области на основе технического задания, которое при автоматизации технологических процессов обычно выдается Заказчиком в словесной форме в виде совокупности сценариев и случаев использования [91], строится структурная схема системы, позволяющая получить общее представление об организации управления, применяемой аппаратуре и интерфейсе объекта управления;

– на этапе анализа на основе технического задания выделяются сущности, каждая из которых называется автоматом (например автомат управления насосом или автомат контроля температуры);

– состояния каждого автомата первоначально определяются по выделенным состояниям объекта управления или его части, а при большом их количестве – по

алгоритму управления, построенному в другой нотации (например в виде схемы алгоритма [51]);

– в автоматы также могут быть введены и другие состояния, связанные, например, с неправильными действиями Оператора;

– каждый автомат при необходимости может быть декомпозирован;

– итеративный процесс анализа может выполняться многократно и завершается созданием перечня автоматов и перечня состояний для каждого из них;

– на этапе проектирования в отличие от традиционного программирования вводится подэтап – кодирование состояний автомата. При этом в каждом автомате для различия состояний применяется многозначный код, в качестве комбинаций которого вводятся десятичные номера состояний;

– автоматы взаимодействуют за счет обмена номерами состояний, вложенности и вызываемости. Они также могут быть одновременно вложенными и вызываемыми;

– строится схема взаимодействия автоматов, отражающая указанные типы взаимодействий. Она формализует систему взаимодействующих автоматов. Эта схема заменяет в предлагаемой технологии диаграмму объектов и частично диаграмму взаимодействий (диаграмму кооперации), которые применяются в объектном моделировании [91];

– входные воздействия разделяются на события, действующие кратковременно, и входные переменные, вводимые путем опроса;

– входные воздействия целесообразно реализовывать в виде входных переменных, а применять события – для сокращения времени реакции системы. При этом одно и то же входное воздействие может быть одновременно представлено и событием и входной переменной;

– прерывания обрабатываются операционной системой и передаются программе в виде сообщений, а после этого обрабатываются как события с помощью соответствующих обработчиков;

– некоторые входные переменные могут формироваться в результате сравнения входных аналоговых сигналов с уставками;

– номера состояний других автоматов, с которыми автомат взаимодействует за счет обмена номерами состояний, также рассматриваются в качестве его входных воздействий;

– все выходные воздействия являются действиями, а не деятельностями;

– группы входных и выходных воздействий связываются с состояниями, выделенными для каждого автомата;

– связи каждого автомата с его "окружением" формализуются схемой связей автомата, предназначенной для полного описания интерфейса автомата. В этой схеме приводятся источники и приемники информации, полные названия всех воздействий и их обозначения, а также информация о том, в какой автомат он вложен и какие автоматы вложены в него;

– имя автомата начинается с символа A, имя события – с символа e (от английского слова event – событие), имя входной переменной – с символа x, имя переменной состояния автомата – с символа y, а имя выходного воздействия – с символа z. После каждого из указанных символов следует номер соответствующего автомата или воздействия;

– система взаимосвязанных автоматов образует системонезависимую (например от операционной системы) часть программы, которая реализует алгоритм функционирования системы управления;

– реализация входных переменных, обработчиков событий, выходных воздействий, вспомогательных модулей и пользовательских интерфейсов образует системозависимую часть программы;

– обработчики событий содержат вызовы функций графов переходов (автоматов) с передачей им соответствующих событий. Функции входных и выходных воздействий вызываются из функций, реализующих автоматы. Функции, образующие вспомогательные модули, вызываются из функций входных и выходных воздействий. Таким образом, автоматы находятся в "центре" структуры программы, создаваемой на основе предлагаемого подхода;

– если составляющая системозависимой части программы спроектирована как автомат (например автомат разбора файла рекомендаций Оператору), то он также может быть введен в схему взаимодействия автоматов;

– если платформа не изменяется, то вспомогательные модули могут быть использованы повторно, как образцы [91];

– запуск автоматов может производиться как из системозависимой части программы (например из обработчиков событий), так и из системонезависимой части;

– вложенные автоматы последовательно запускаются с передачей "текущего" события в соответствии с путем в схеме взаимодействия автоматов, определяемым их состояниями в момент запуска головного автомата. При этом последовательность запуска и завершения работы автоматов напоминает алгоритм поиска в глубину [157];

– вызываемые автоматы запускаются из выходных воздействий с передачей соответствующих "внутренних" событий;

– автоматы могут запускаться однократно с передачей какого-либо события или многократно (в цикле) с передачей одного и того же события;

– при реализации системы учтено, что функции, реализующие автоматы, нереентерабельны (не допускают повторного запуска до их завершения);

– каждый автомат при запуске выполняет не более одного перехода;

– после обработки очередного события автомат сохраняет свое состояние и "засыпает" до появления следующего события;

– дуги и петли графов переходов помечаются произвольными логическими формулами, которые могут содержать входные переменные и предикаты, проверяющие номера состояний других автоматов и номера событий;

– дуги и петли кроме условий переходов могут содержать список последовательно выполняемых выходных воздействий;

– вершины в графах переходов практически всегда являются устойчивыми и содержат петли. Если на петле не выполняются выходные воздействия, то она умалчивается. В противном случае в явном виде изображаются одна или несколько петель, каждая из которых помечена, по крайней мере, выходными воздействиями;

– вершина графа переходов может содержать список последовательно запускаемых вложенных автоматов и список последовательно выполняемых выходных воздействий;

– для обобщения "одинаковых" исходящих дуг в каждом графе переходов допускается объединение вершин в группы. Также допускается слияние входящих в вершину дуг в одну линию;

– каждый граф переходов проверяется на достижимость, непротиворечивость, полноту и отсутствие генерирующих контуров;

– этап завершается построением графа переходов для каждого автомата, совокупность которых образует систему взаимосвязанных автоматов;

– на этапе реализации строится программа, в которой графы переходов, входные переменные, обработчики событий и выходные воздействия выполняются в виде

функций. Кроме того, программа содержит вспомогательные модули (например, модуль управления таймерами);

– для хранения номера состояния автомата (различения состояний) используется одна внутренняя переменная. Для различения изменения состояния применяется вторая переменная, носящая вспомогательный характер;

– разработан универсальный алгоритм программной реализации иерархии графов переходов с произвольным их количеством и произвольным уровнем вложенности;

– каждый граф переходов формально и изоморфно реализуется отдельной функцией (подпрограммой), создаваемой по шаблону, содержащему две конструкции switch и оператор if. Первая конструкция switch вызывает вложенные автоматы и реализует переходы и действия на дугах и петлях. Оператор if проверяет изменилось ли состояние, и если оно изменилось, вторая конструкция switch активизирует вложенные автоматы и реализует действия в новой вершине;

– после реализации графа переходов текст подпрограммы должен корректироваться для обеспечения бесповторности опроса входных переменных, помечающих дуги, исходящие из одного состояния. Таким образом, решается проблема "риска" [8];

– каждая входная переменная и каждое выходное воздействие также реализуется функцией, что и позволяет применять SWITCH-технологию не только для решения задач логического управления;

– имена функций и переменных, используемых при реализации автоматов, совпадают с обозначениями, применяемыми в схемах связей автоматов и графах переходов. Например, переменная, в которой хранится номер произошедшего события, имеет имя *e*;

– все функции, реализующие входные переменные, записываются в порядке возрастания их номеров в один файл, а реализующие выходные воздействия – в другой;

– функции, реализующие автоматы, входные переменные и выходные воздействия, содержат вызовы функций для обеспечения протоколирования;

– этап завершается построением структурной схемы разработанного программного обеспечения, отражающей взаимодействие его частей. Эта схема может включать схему взаимодействия автоматов, которая при этом отдельно не выпускается;

– на этапе отладки обеспечена возможность одновременной индикации значений переменных состояний всех автоматов на одном экране;

– на этапе сертификации за счет введения в функции автоматов, входных и выходных воздействий вызовов функций протоколирования обеспечено автоматическое ведение протокола. В нем указываются события, запуск автоматов, их состояния в момент запуска, переходы в новые состояния, завершение работы автоматов, значения входных переменных, выходные воздействия и время начала выполнения каждого из них. Кроме "полного" протокола также автоматически строится "короткий" протокол, в котором фиксируются только события и инициируемые ими выходные воздействия, интересующие Заказчика;

– сообщения в "полном" протоколе о запуске и завершении реализации каждого автомата играют роль скобок, логически выделяющих разные уровни вложенности автоматов;

– на этапе документирования для точного оформления результатов проектирования и разработки программы создается и сдается в архив документация (по крайней мере в электронном виде), имеющая как минимум следующую комплектность: структурная схема системы; схема разработанного программного обеспечения; распечатки экранов пользовательских интерфейсов; перечни событий, входных переменных и выходных воздействий; диаграмма взаимодействия автоматов; описание

нотации, используемой в графах переходов; шаблон для реализации графов переходов смешанных автоматов произвольного уровня вложенности; для каждого автомата: словесное описание (фрагмент технического задания), рассматриваемое в качестве комментария, схема связей автомата, граф переходов и исходный текст функции, реализующей автомат; описания алгоритмов, например в виде графа переходов, и исходные тексты вспомогательных модулей и функций, реализующих входные переменные, обработчики событий и выходные воздействия; протоколы для сертификации программы, выполняющие роль контрольных примеров [158]; руководство программиста; руководство пользователя;

– изложенный вариант технологии может использоваться и при построении модели объекта управления, для которой должен создаваться аналогичный комплект документации;

– после этого при появлении любых изменений, возникающих в ходе дальнейших этапов жизненного цикла программы, весь комплект документации (по завершении каждого этапа) должен корректироваться [159]. Для этого составляется перечень исполняемых модулей программы, в котором для каждого из них указывается значение циклической контрольной суммы, отражающей любое изменение в модуле. При этом Контролер по документации должен знать значение этой суммы для исходного файла, и после завершения каждого этапа при любом изменении указанного значения требовать представления извещения на выполненную модификацию, по которому документация должна быть комплектно откорректирована и сдана в архив.

Излагаемый вариант технологии обладает следующими достоинствами:

– в отличие от объектного моделирования [91, 113], во-первых, построение всех основных моделей основано на применении только автоматной терминологии, а во-вторых, используется динамическая модель только одного типа – система взаимосвязанных графов переходов;

– применение такой динамической модели позволяет эффективно описывать и реализовывать задачи рассматриваемого класса даже при большой их размерности. Применение графов переходов в качестве языка спецификаций алгоритмов делает обозримым даже весьма сложное поведение программы и позволяет легко вносить изменения как в спецификацию, так и в ее реализацию;

– совместное рассмотрение схемы связей автомата и его графа переходов позволяет понимать этот граф, а совместное рассмотрение этого графа и изоморфной ему подпрограммы позволяет понимать эту подпрограмму;

– подробное документирование проекта создания программного обеспечения позволяет при необходимости вносить изменения в него через длительный срок после его выпуска и даже другими Специалистами;

– без использования объектно-ориентированного подхода программа четко разделяется на две части – системонезависимую и системозависимую;

– при проектировании системонезависимой части программы детали реализации входных и выходных воздействий скрыты. Они раскрываются только при реализации системозависимой части программы;

– этапы проектирования и реализации системонезависимой части программы полностью разделены;

– реализация входных переменных и выходных воздействий в виде функций обеспечивает: их протоколирование, простоту перехода от одних типов источников и приемников информации к другим, наличие действующего макета программы [158] в любой момент времени после начала реализации системозависимой части;

– упорядоченное хранение функций, реализующих входные переменные и выходные воздействия, упрощает внесение изменений;

– для кодирования любого числа состояний автомата используется только одна внутренняя переменная, что обеспечивает наблюдаемость поведения автомата за счет "слежения" за изменениями значений только этой переменной. Для системы из N автоматов "слежение" выполняется по N многозначным переменным, значения каждой из которых выводятся на "отладочный" экран, представленный в форме, определяемой схемой взаимодействия автоматов;

– каждый граф переходов формально и изоморфно реализуется по шаблону, что при необходимости позволяет решить обратную задачу – однозначно восстановить граф переходов по этой подпрограмме;

– системонезависимая часть программы имеет регулярную структуру и, следовательно, легко читается и корректируется;

– системонезависимая часть программы зависит только от наличия компилятора или интерпретатора выбранного языка программирования на используемой платформе. При смене аппаратуры или переносе программы под другую операционную систему необходимо изменить только системозависимую часть;

– автоматическое ведение протокола в терминах спецификации обеспечивает возможность сертификации программы. При этом демонстрируется соответствие функционирования программы "поведению" системы взаимосвязанных графов переходов для рассматриваемых событий при выбранных значениях входных переменных. Это достигается за счет сопоставления "полного" протокола со спецификацией. Совокупность "полных" протоколов обеспечивает возможность сертификации программы в целом. Для сертификации в терминах, понятных Заказчику, могут применяться "короткие" протоколы, которые могут использоваться также и в качестве фрагментов методики проверки функционирования системы. При этом отметим, что из двух групп понятий "объект – алгоритм" и "алгоритм – программа" сертификация обычно связывается только со второй группой понятий;

– "короткий" протокол позволяет определить наличие ошибки в выдаче выходных воздействий, а "полный" – определить автомат, который при этом необходимо откорректировать. Поэтому "короткие" протоколы могут быть названы "проверяющими", а "полные" – "диагностирующими";

– возможность автоматического получения "полных" протоколов в терминах автоматов показывает, что система взаимосвязанных графов переходов, используемая для спецификации алгоритмов, является не "картинкой", а математической моделью;

– несмотря на достаточно высокую трудоемкость проведения "подробной" сертификации при применении предлагаемых протоколов, этот способ существенно более практичен, чем другие подходы к построению качественных программ, которые изложены, например в [160, 161];

– порождаемый некоторыми событиями протокол или его часть является соответствующим сценарием. Таким образом, сценарий [64] строится автоматически при анализе программы, а не вручную при ее синтезе, как это предлагается делать при других подходах [91, 113]. Ручное построение всей совокупности сценариев и формальный синтез системонезависимой части программы по ним для задач со сложной логикой практически не осуществимы;

– излагаемый подход не исключает интерактивной отладки и сертификации;

– поведение системы взаимосвязанных автоматов является разновидностью коллективного поведения автоматов [162] и может использоваться при построении "многоагентных систем, состоящих из реактивных агентов" [163].

Разработка системы управления дизель-генератором, выполненная с помощью предлагаемого варианта технологии, подтвердила мнение Ф. Брукса [158] о том, что время, затрачиваемое на проектирование алгоритма, при котором строятся не кар-

тинки, а математические модели, значительно превышает время, затрачиваемое на написание системонезависимой части программы, изоморфной спроектированному алгоритму, а также мнение Д. Воаса [164] о CASE-средствах, которые сами по себе мало что убыстряют, а "практика их применения показала только то, что из некорректных картинок можно получить некорректный код".

Текст подпрограммы, построенной с помощью предложенного варианта технологии, и "полный" протокол ее работы, проверяющий все переходы автомата, реализующего алгоритм управления объектом "тулбар", приведен в п. 3.2 Приложения к работе [54].

6. Применение конечных автоматов при программировании схем программируемой логики

Подход, аналогичный изложенному для программной реализации алгоритмов, широко используется в настоящее время для настройки схем программируемой логики [165], также называемой программированием [166]. При этом разработаны языки описания аппаратуры (Hardware Description Language (HDL) в англоязычной литературе), которые, в частности, позволяют конвертировать текст программы в функциональную схему и наоборот. Описание, выполненное на одном из таких языков, компилируется в схему [167]. Среди языков этого класса отметим языки VHDL [168] и AHDL [169].

Например, в языке AHDL для реализации условной логики (Conditional Logic) включены конструкции if then и case, последняя из которых является аналогом конструкции switch языка СИ. При этом одно из "золотых" правил, приведенных в [169], состоит в применении везде, где это возможно, конструкции case вместо вложенных конструкций if then.

Это правило используется для реализации последовательностной логики (Sequential Logic), описываемой в терминах состояний с помощью конечных автоматов (State Machine). При этом автоматы с многозначным кодированием состояний (этот термин в [168] не применяется) задаются в виде диаграмм состояний (State Diagram). В [169] приводятся примеры реализации автоматов с синхронными выходами (State Machines with Synchronous Outputs), соответствующих автоматам Мура, и автоматов с асинхронными выходами (State Machines with Asynchronous Outputs), соответствующих автоматам Мили.

Дальнейшее развитие автоматного подхода для настройки схем программируемой логики состоит в визуализации диаграмм состояний, которая осуществляется с помощью пакетов типа "State CAD", один из которых описан в [170].

Таким образом, технология проектирования алгоритмического и программного обеспечения, изложенная в разд. 2 и 5, дополняет известные подходы к настройке схем программируемой логики, что позволяет с единых позиций проводить аппаратурную [171] и программную реализации алгоритмов логического управления, так же как это имеет место в технологии "Co-Design", предназначенной для совместной разработки аппаратурных и программных средств с помощью формализованных методов, в которой в качестве языка описания высшего уровня абстракции применяются конечные автоматы [172].

7. Лауреаты премии Тьюринга о применении понятия "состояние" в программировании

После издания [8] автор познакомился с работами [173, 174]. Некоторые из особенностей излагаемого подхода, базирующегося на парадигме конечных автоматов, являющейся одной из парадигм программирования, рассмотренных Р. Флой-

дом [173], совпадают с предложениями, высказанными лауреатами премии Тьюринга, самим А. Тьюрингом [183] и Дж. фон Нейманом [184].

Так, А. Дж. Перлис в 1966 г. [173] предложил в описания языка, среды и правил вычислений включить состояния, которые могут подвергаться мониторингу во время исполнения, позволяя диагностировать программы, не нарушая их целостности. При этом под мониторингом он понимал распределенное управление, более известное в настоящее время под названием "объектно-ориентированное программирование".

В этом же году Э. Дейкстра [174] предложил ввести так называемые переменные состояния, с помощью которых можно описывать состояния системы в любой момент времени, и ввел для этих целей целочисленные переменные. При этом им были поставлены вопросы о том, какие состояния должны вводится, как много значений должны иметь переменные состояния и что эти значения должны означать. Он предложил сначала определять набор подходящих состояний, а лишь затем строить программу. Он также предложил сопоставлять процессы с переменными состояниями и связывать процессы через эти переменные. По мнению Э. Дейкстры, диаграммы состояний могут оказаться мощнм средством для проверки программ. Все это обеспечивает поддержку его идеи, состоящей в том, что программы должны быть с самого начала составлены правильно, а не отлаживаться до тех пор, пока они не станут правильными.

В 1977 г. Дж. Бэкус [173] отметил, что в языках программирования для компьютера фон Неймана семантика тесно сплетается с переходами между состояниями и их недостатком является то, что "всякая подробность вычислений изменяет состояние. Вследствие такой тесной связи семантики с состояниями всякая деталь каждого свойства должна быть встроена в состояние и в его правила переходов". При этом он указал на необходимость того, чтобы вычислительные системы обладали свойством "исторической чувствительности", и отметил, что "система не может быть исторически чувствительной (допускать влияние выполнения одной программы на поведение следующей программы), если она не обладает некоторым состоянием, которое первая программа может изменять, а вторая воспринимать. Поэтому исторически чувствительная модель вычислительной системы должна обладать семантикой смены состояний". Он предложил рассматривать состояние в целом, а его изменение производить только после "большого" вычисления.

8. Заключение

Автор надеется, что изложенный подход, использующий элементы бихевиоризма, когнитивной психологии [57] и гештальт-психологии [41], позволит повысить качество проектирования и реализации программ логического управления ответственными технологическими объектами, в том числе и за счет автопрограммирования, при котором человек, знающий объект, сам выполняет алгоритмизацию и программирование, так как для рассматриваемого класса задач предлагаемая система обозначений является "естественным продолжением образа мышления, а не чуждого ему формализма" (Н. Вирт в [173]). Графы переходов в качестве языка алгоритмизации, как и любой "язык, на котором записывается решение задачи, напрямую влияют на ход мыслей человека, заставляя его рассматривать задачу под определенным углом" [37], определяя "дисциплину мышления" [20]. В настоящее время считается [77], что графы переходов позволяют наиболее естественным образом описывать процессы, протекающие в реальном мире, дополняя объектный подход, являющийся "естественным способом размышления о мире" [175].

Если процедурное программирование ориентировано на выбор и действия, объектное программирование – на объекты [175], событийное программирование – на

события, то предлагаемое в [8] автоматное программирование – на автоматы, включающие состояния, входные и выходные воздействия.

При этом отметим, что по "данным Киевского представительства фирмы "Шнайдер Электрик" в настоящее время только 4% украинских пользователей ее ПЛК используют язык "Графсет", входящий в стандарт IEC 1131-3, что, видимо, связано с инерцией мышления у программистов, которые в большинстве случаев автоматически переносят стиль программирования на языках для персональных компьютеров на программирование для ПЛК" [9].

Однако, по мнению автора, указанная ситуация связана не только с особенностями пользователей, но и с методическим обеспечением применения этого языка и его реализацией. Так, например, в [176] сформулирована одна из задач логического управления и для нее построена диаграмма "Графсет", которая, однако, является "картинкой", а не математической моделью. Это исключает возможность формального и изоморфного перехода от нее к программе и применение диаграммы в качестве сертификационного теста. Удивительным является то, что эта диаграмма в [176] не используется для написания программы на языке "Графсет", а по ее "мотивам" эвристически строятся лестничная схема и программа на языке инструкций. Кроме того, как было отмечено в [8], графы переходов в системах логического управления обычно имеют многочисленные возвраты назад, что делает реализующие их диаграммы "Графсет" ненаглядными. Небольшое число двоичных переменных, выделяемых в ПЛК для кодирования вершин в этих диаграммах, резко ограничивает допустимое число вершин в реализуемых таким образом графах переходов.

По мнению автора, в рамках рассмотренной технологии программирование ПЛК наиболее целесообразно выполнять на языке "структурированный текст" ("Structured Text") [1], особенно для тех ПЛК, в которых этот язык содержит конструкцию, аналогичную конструкции switch языка СИ.

Таким образом, в настоящее время SWITCH-технология использована при построении систем управления применительно ко всем трем типам управляющих вычислительных устройств, перечисленных в начале работы, и показала свою эффективность.

Из изложенного следует, что для рассматриваемого класса задач SWITCH-технология обеспечивает построение "добротных" [177] программ, подтверждая тем самым высказывание: "то, что не специфицировано формально, не может быть проверено, а то, что не может быть проверено, не может быть безошибочным" [38].

"Сложность служит причиной трудности перечисления, а тем более понимания, всех возможных состояний программы, а отсюда возникает ее ненадежность. Сложность служит также источником невизуализируемых состояний, в которых нарушается система защит" [158].

Описанная технология основана на априорном задании состояний и их визуализации, и поэтому хочется надеяться, что она, обладая минимализмом [178], по крайней мере для систем логического управления и "реактивных" систем, является приближением к "серебряной пуле" [158] в части создания качественных программ, тем более что Ф. Брукс в этой работе отозвался благосклонно только о подходе Д. Харела, сравнение с которым выполнено выше.

Эта технология может рассматриваться как одно из направлений выхода из "кризиса линейного мышления" [179], другое из которых, например, связано с заменой линейного текста гипертекстом.

Применение изложенной технологии может быть особенно важным для обеспечения требований международного стандарта IEC 880 "Программное обеспечение ЭВМ, использующихся в системах эксплуатационной надежности атомных электростанций (АЭС)", который регламентирует в том числе и процесс разработки и контроля за созданием программного обеспечения для систем управления ядерными

энергетическими установками, а также порядок внесения в него изменений [180]. Применение в предлагаемой парадигме понятия "автомат" в качестве центрального понятия соответствует его месту в теории управления, что принципиально отличает ее от других парадигм программирования.

СПИСОК ЛИТЕРАТУРЫ

1. International standard IEC 1131-3. Programmable controllers. Part 3. Programming languages. International Electrotechnical Commission. 1993.
2. SIMATIC. Simatic S7/M7/C7. Programmable controllers. SIEMENS. Catalog ST 70. 1996.
3. TSX T607. Programming terminal. User's manual. Telemecanique. 1987.
4. Modicon catalog & specifier's guide. Modicon. AEG Schneider Automation. 1995.
5. Programmable controller. MELSEC – A. Programming manual. Type ACPU. Common instructions. Mitsubishi Electric.
6. ABB Procontic T200. Mid-range automation systems using modern technology. Asea Brown Boveri. 1994.
7. ET-PDS. Software for programmable logic controllers. Tosiba International (Europe) Ltd. 1995.
8. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
9. *Гузик И. М.* Стандарт МЭК 1131: язык GRAFCET – знакомство поближе // Schneider Automation Club. 1999. № 6.
10. *Лавров С. С.* Лекции по теории программирования. Учебное пособие. СПб.: СПбГТУ, Нестор, 1999.
11. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.: Бином; СПб.: Невский диалект, 1998.
12. *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования. М.: Мир, 1999.
13. *Клир Д.* Абстрактное понятие системы как методологическое средство / Исследования по общей теории систем. М.: Прогресс, 1969.
14. *Кузнецов Б. П.* Структура и сложность модулей циклических программ // Автоматика и телемеханика. 1999. № 2.
15. *Odell J. J.* Advanced object-oriented analysis & design using UML. NY: SIGS Books. 1998.
16. *Таль А. А., Айзерман М. А., Розоноэр Л. И. и др.* Логика. Автоматы. Алгоритмы. М.: Физматгиз, 1963.
17. *Шалыто А. А.* Использование граф-схем алгоритмов и графов переходов при программной реализации алгоритмов логического управления // Автоматика и телемеханика. 1996. № 6, 7.
18. *Глушков В. М.* Синтез цифровых автоматов. М.: Физматгиз, 1962.
19. *Yourdon E., Argila C.* Case studies in object-oriented analysis & design. NJ: Yourdon Press, 1997. (Йордон Э., Аргила К. Структурные модели в объектно-ориентированном анализе и проектировании. М.: Лори, 1999).
20. *Росс Д.* Структурный анализ: язык для передачи понимания / Требования и спецификации в разработке программ. М.: Мир, 1984.
21. *Chow T. S.* Testing software design modeled by finite state machines // IEEE Trans. Soft. Eng. 1978. No. 3.
22. *King D. et al.* On object state testing / Proceeding The Eighteenth Annual Inter. Comp. Software & Applications Conf. Los Alamitos: IEEE Computer Sosiety Press, 1993.
23. *Turner C. D., Robson D. J.* The state-based testing of object-oriented programs / Conf. on Software Maintenance. Los Alamitos: IEEE Computer Sosiety Press, 1993.

24. *Jorgenson P., Erickson C.* Object-oriented integration testing // Communications of the ACM. 1994. No. 9.

25. *Bunder R. V.* The FREE-flow graph: implementation-based testing of objects using state-determined flows / Proc. 8th Annual Software Quality Week. San Francisco: Software Research Inc., 1995.

26. *Бурдонов И. Б., Косачев А. С., Кулямин В. В.* Использование конечных автоматов для тестирования программ // Программирование. 2000. № 2.

27. *Kurshan R. P.* Computer-aided verification of coordinated prosesses – the automata-theoretic approach. Prenceton: Prenceton University Press, 1994.

28. *Kurshan R. P.* Program verification // Notices of the ACM. 2000. No. 5.

29. *Шалыто А. А.* Реализация алгоритмов логического управления программами на языке функциональных блоков // Промышленные АСУ и контроллеры. 2000. № 4.

30. *Martin R. C.* Designing object-oriented C++ applications using the Booch method. NJ: Prentice-Hall, 1995.

31. *Пригожин И., Стенгерс И.* Порядок из хаоса. М.: Эдиториал УРСС, 2000.

32. *Трахтенброт Б. А., Бардзинь Я. М.* Конечные автоматы. Поведение и синтез. М.: Наука, 1970.

33. *Гилл А.* Введение в теорию конечных автоматов. М.: Наука, 1966.

34. *Фридман А., Менон П.* Теория и проектирование переключательных схем. М.: Мир, 1978.

35. *Канер С., Фолк Д., Енг Кек Нгуен.* Тестирование программного обеспечения. Киев: DiaSoft, 2000.

36. *Руднев В. В.* Система взаимосвязанных графов и моделирование дискретных процессов // Автоматика и телемеханика. 1984. № 9.

37. *Бадд Т.* Объектно-ориентированное программирование в действии. СПб.: Питер, 1997.

38. *Зайцев С. С.* Описание и реализация протоколов сетей ЭВМ. М.: Наука, 1989.

39. *Шалыто А. А., Туккель Н. И.* SWITCH-технология – автоматный подход к созданию программного обеспечения "реактивных" систем / Телематика 2000. Тез. докл. международной научно-метод. конф. СПб.: СПбГИТМО (ТУ), 2000.

40. *Калянов Г. Н.* CASE. Структурный анализ (автоматизация и применение). М.: Лори, 1996.

41. *Шульц Д., Шульц С.* История современной психологии. СПб.: Евразия, 1998.

42. Functinal description. Warm-up & prelubrication logic. Generator control unit. Severnaya hull N431. Norcontrol. 1993.

43. Project 15640. AS 21. DG 1. Control. АМИЕ. 95564.12М. St. Petersburg. ASS "Avrora". 1991.

44. Система управления турбокомпрессорным агрегатом "Ларина". Техническое описание. Приложение 1. АМИЕ. 421417.010 ТО.01. СПб.: НПО "Аврора", 1998.

45. Autolog 32. Руководство пользователя. FF-Automation OY.

46. *Баглюк Ю. В., Шалыто А. А.* Программируемые логические контроллеры "Autolog". 124 примера программ на языке "ALPro", реализующих алгоритмы логического управления. СПб.: FF-Automation, 1999. На русском и английском языках.

47. Транслятор "CF – ALPro" для программирования контроллеров типа "Autolog". Руководство пользователя. NZF.TR 1.1РП (на русском языке); NZF.TR 2.1РП (на английском языке). FF-Automation, 1999.

48. *Литов Д. В.* Автомат или машина Тьюринга // Мир ПК. 1999. № 3.

49. *Баранов С. И.* Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.

50. *Любченко В. С.* Мы выбираем, нас выбирают... (к проблеме выбора алгоритмической модели // Мир ПК. 1999. № 3.

51. *Затуливетер Ю. С., Халатян Т. Г.* Синтез общих алгоритмов по демонстрациям частных примеров (автоматная модель обобщения по примерам). М.: Ин-т проблем управления, 1997.

52. *Матвеев В. И.* Windows CE – новый этап в развитии PLC // Приборы и системы управления. 1999. № 3.

53. *Туккель Н. И., Шалыто А. А.* Сравнение событийного и автоматного подходов к программированию задач логического управления / Телематика 99. Тез. докл. Всерос. научно-метод. конф. СПб.: СПбГИТМО (ТУ), 1999.

54. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000.

55. *Tukel N.* Programming with use of a SWITCH-technology // Euroxchange. Special student's issue. ISA. 1999. No. 2.

56. *Шалыто А. А.* Технология программной реализации алгоритмов логического управления как средство повышения живучести // Проблемы обеспечения живучести кораблей и судов. Тез. докл. научно-техн. конф. СПб.: НТО им. акад. А. Н. Крылова, 1992.

57. *Shalyto A. A.* Cognitive properties of hierarchical representations of complex logical structures // Archetecturs for semiotic modeling and situation analysis in large complex systems. 10th IEEE Intern. symposium on intellegent control. Monterey, California, 1995.

58. *Шалыто А. А., Антипов В. В.* Алгоритмизация и программирование задач логического управления техническими средствами. СПб.: Моринтех, 1996. Книжное издание и лазерный диск.

59. *Bagljuk Y. V., Shalyto A. A.* SWITCH-technology. Algorithmic and programming methods in solution the logic control problems of shipping equipment // Intern. conference on informatics and control. ICI & C'97. Proceedings. Vol. 1. St. Petersburg, 1997.

60. *Кузнецов О. П., Макаревский А. Я., Марковский А. В. и др.* Ярус – язык описания работы сложных автоматов // Автоматика и телемеханика. 1972. № 6, 7.

61. *Кузнецов О. П.* Графы логических автоматов и их преобразования // Автоматика и телемеханика. 1975. № 9.

62. *Кузнецов О. П., Шипилина Л. Б., Григорян А. К. и др.* Проблемы разработки языков логического программирования и их реализация на микро-ЭВМ (на примере языка "Ярус – 2") //Автоматика и телемеханика. 1985. № 6.

63. *Любашин А.* Промышленные и встраиваемые системы. На "стандартном" пути // PC WEEK. 2000. № 15.

64. ADAM-5510/P31 / Все необходимое для автоматизации на базе PC. Advantech. 1999. Т. 91.

65. Programmable controllers fundamentals. Allen-Bradley Inc., 1985.

66. Micro Mentor: understanding and applying micro programmable controllers. Allen-Bradley Inc., 1995.

67. *Jones C. T., Bryan L. A.* Programmable controllers concepts and applications. International Programmable Controllers Inc., 1983.

68. *Gilbert R. A., Llewellyn J. A.* Programmable controllers – practices and concepts. Industrial Training Corporation, 1985.

69. *Lloyd M.* Grafset – graphical functional charts programming for programmable controllers // Measurement & Control Magazine. 1987. No. 9.

70. *Bryan L. A., Bryan E. A.* Programmable controllers: theory and implementation. Industrial Text Corp., 1988.

71. *Webb J. W., Reis R. A.* Programmable logic controllers: principles and applications. NJ: Prentice-Hall, 1995.

72. *Wisnosky D. E.* SoftLogic: overcoming funnel vision. Wizdom Controls Inc., 1996.

73. *Highes T. A.* Programmable controllers. Instrument Society of America (ISA) Publications, 1997.

74. *Верников Г.* Основы обследования деятельности организаций. Стандарт IDEF3 // READ.ME. 2000. № 2.

75. *Шалыто А. А.* Программная реализация управляющих автоматов // Судостроит. пром-сть. Сер. Автоматика и телемеханика. 1991. Вып. 13.

76. *Боуэн Д., Хинчи М.* Десять заповедей формальных методов // Мир ПК. 1997. № 9, 10.

77. Series 90-70. State logic control system. User's manual. NA: GE Fanuc Automation. 1998.

78. Промышленные контроллеры фирмы "Matsushita". Matsushita Automation Controls.

79. Программное обеспечение для программируемых контроллеров и визуализации. Festo Cybernetic.

80. *Мартынюк В. В.* Об анализе графа переходов для операторной схемы // Журн. вычисл. математики и мат. физики. 1965. № 2.

81. *Карпов Ю. Г.* Основы построения компиляторов. Учебное пособие. Л.: Изд-во ЛПИ, 1982.

82. *Касьянов В. Н., Поттосин И. В.* Методы построения трансляторов. Новосибирск: Наука, 1986.

83. *Байцер Б.* Архитектура вычислительных комплексов. Т. 1. М.: Мир, 1974.

84. *Буч Г.* Объектно-ориентированное проектирование с примерами применения. Киев: Диалектика; М.: ИВК, 1992.

85. *Страуструп Б.* Язык программирования Си++. М.: Радио и связь, 1991.

86. *Любченко В. С.* Новые песни о главном (римейк для программистов) // Мир ПК. 1998. № 6, 7.

87. *Shlaer S., Mellor S.* Object Lifecycles: Modeling the World in State. NJ: Prentice-Hall, 1992. (Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. Киев: Диалектика, 1993).

88. *Harel D.* Statecharts: A visial formalism for complex systems // Sci. Comput. Program. 1987. Vol. 8.

89. *Harel D. et al.* STATEMATE: A working environment for the development of complex reactive systems // IEEE Trans. Eng. 1990. No. 4.

90. *Douglass B. P.* UML Statecharts. MA: I-Logix Inc., 1998.

91. *Booch G., Rumbaugh J., Jacobson I.* The Unified Modeling Language. User guide. MA: Addison-Wesley, 1998. (Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК, 2000).

92. Unified Modeling Language (UML). Version 1.0. CA: Rational Software Corp., 1997.

93. *Harel D., Pnueli A.* On the development of reactive systems / Logic and model of computer systems. Ed. K.R. Apt. NY: Springer-Verlag, 1985.

94. *Harel D. et al.* On formal semantics of software / Pros. 2nd IEEE Symp. Logic in Computer Science. NY: IEEE Press, 1987.

95. *Coleman D., Hayes E., Bear S.* Introducing objectcharts, or how to use statecharts in object oriented design // IEEE Trans. Soft. Eng. 1992. No. 1.

96. *Harel D., Naamad A.* The STATEMATE semantics of statecharts // ACM Trans. Soft. Eng. Metodology. 1996. No. 10.

97. *Harel D., Gery E.* Executable object modeling with statecharts // Computer. 1997. No. 7.

98. *Harel D., Politi M.* Modeling reactive systems with statecharts. NY: McGraw-Hill, 1998.

99. *Douglass B. P.* Real-time UML: Developing efficient objects for embedded systems. MA: Addison-Wesley, 1998.

100. *Douglass B. P.* Doing hard time: Using object oriented programming and software patterns in real time applications. MA: Addison-Wesley, 1998.

101. *Карпов Ю. Г.* Теория алгоритмов и автоматов. Курс лекций. СПБ.: СПбГТУ, Нестор, 1998.

102. Information processing systems. Open systems interaction. ESTELLE: a formal description technique based on an extended state transition model // International standard. ISO 9074, 1989.

103. *Bochman G. V.* Finite state description of communication protocols // Comput. Network Protocol Sympl. Liege, 1978, V. 2.

104. *Dantine A.* Protocol representation with finite state models // IEEE Trans. on Commun. 1980. No. 4.

105. *Brand D., Zafiropulo P.* On communicating finite state machines // Journal ACM. 1983. No. 2.

106. CCITT Recomendation Z.100: CCITT Specification and description language (SDL) // COM X-R 26, ITU General Secretariat. Geneva, 1992.

107. *Braek R., Haugen F.* Engineering real time systems. NJ: Prentice-Hall, 1993.

108. *Гольштейн Б. С.* Сигнализация в сетях связи. М.: Радио и связь, 1997.

109. *Иванов А., Кознов Д., Мурашова Т.* Поведенческая модель RTST++ // Записки семинара кафедры системного программирования. CASE-средства RTST++. Вып. 1. СПб.: Изд-во СПбГУ, 1998.

110. *Парфенов В. В., Терехов А. Н.* RTST-технология программирования встроенных систем реального времени // Системная информатика. Вып. 5. Новосибирск: Наука, 1997.

111. *Терехов А. Н.* RTST-технология программирования встроенных систем реального времени // Записки семинара кафедры системного программирования. CASE - средства RTST++. Вып. 1. СПб.: Изд-во СПбГУ, 1998.

112. *Долгов П., Иванов А., Терехов А. и др.* Объектно-ориентированное расширение технологии RTST // Записки семинара кафедры системного программирования. CASE – средства RTST++. Вып. 1. СПб.: Изд-во СПбГУ, 1998.

113. *Терехов А. Н., Романовский К. Ю., Кознов Д. В. и др.* REAL: Методология и CASE-средство разработки информационных систем и программного обеспечения систем реального времени // Программирование. 1999. № 5.

114. *Кознов Д. В.* Конечный автомат – основа для визуальных представлений поведения объектов / Объектно-ориентированное визуальное моделирование. СПб.: Изд-во СПбГУ, 1999.

115. Microsoft. Решения'99. Microsoft Corp. 1999. Вып. 7.

116. *Cook S., Daniels J.* Designing object systems. Object-oriented modeling with syntropy. NJ: Prentice-Hall, 1994.

117. *Rambaugh J., Blaha M., Premerlani W. et al.* Object-oriented modeling and design. NJ: Printice-Hall. 1991.

118. *Jacobson I.* Object-oriented software engineering: A use case driven approach. MA: Addison-Wesley, 1992.

119. xjCharts. Release 2.0. User's Manual. Experimental Object Technologies. 1999.

120. STATEFLOW for use with Simmulink. User's guide. Version 1. MA: Math Works, Inc. 1998.

121. MATLAB. The language of technical computing. Version 5.2. MA: Math Works, Inc. 1998.

122. *Зюбин В. Е.* К пятилетию стандарта IEC 1131-3. Итоги и прогнозы // Приборы и системы управления. 1999. № 1.

123. *Герр Р.* Новый поворот // PC Magazine / Russian Edition. 1998. № 10.

124. *Губанов Ю. А., Залманов С. З., Кузнецов Б. П.* Управление автоматическими выключателями корабельной электроэнергетической системы / Третья международная конф. по морским интеллектуальным технологиям "Моринтех-99". СПб.: Моринтех, 1999, Т. 3.

125. *Менделевич В. А.* Непроцедурные языки – новое поколение средств разработки АСУ ТП // Промышленные АСУ и контроллеры. 2000. № 1.

126. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы. Построение и анализ. М.: МЦНТО, 1999.

127. *Odell J. J.* Approaches to finite-state machine modeling // Journal of obiect-oriented programming. 1995. № 1.

128. *Heizinger T., Manna Z., Pnueli A.* Timed transition systems. Technical report TR 92-1263. Dept. of Computer Science. Cornell University. 1992.

129. *Ward P., Mellor S.* Structured techniques for real-time systems. NJ: Yourdon Press – Prentice-Hall, 1985.

130. *Hatley D., Pirbhai I.* Strategies for real-time system specification. NY: Dorset House, 1987.

131. *Drysinsky D.* Visual programming. Better state. Product overview. Cupertino. California. R-Active Concepts, 1993.

132. *Selic B.* An efficient object-oriented variation of statecharts formalism for distributed real-time systems // CHDL'93: IFIP Conf. on hardware description languages and their applications. Ottawa, 1993.

133. *Selic B., Gullekson G., Ward P.* Real-time object-oriented modeling. NY: John Wiley & Sons Inc., 1994.

134. *Borshchev A. V., Karpov Y. G., Roudakov V. V.* COVERS – A tool for the design of real-time concurrent systems // Parallel Computing Technologies. Proceedings of the 3rd Inter. Conf. PACT-95. Lecture Note in Computer Science. 1995. No. 964.

135. *Booch G., Rambaugh J.* Unified method for object-oriented development. Documentation Set. Version 0.8. Rational Software Corp., 1996.

136. *Сонкин В. Л., Мартинов Г. М., Любимов А. Б.* Интерпретация диалога в Windows-интерфейсе систем управления // Приборы и системы управления. 1998. № 12.

137. *Ran A. S.* Modeling states as classes / Proc. of the tools USA 94. Ed. M. Singh, B.Meyer. NJ: Prentice-Hall, 1994.

138. *Ran A. S.* Patterns of events / Pattern languages of programm design. Ed. J. O. Coplien, D. C. Schmidt. MA: Addison-Wesley, 1995.

139. *Ilgum K., Kemmerer R., Porras P.* State transition analysis: a rule-based intrusion detection approach // IEEE Trans. on Software Eng. 1995. No. 3.

140. *Corbett J. C.* Evaluating deadlock detection methods for concurrent software // IEEE Trans. on Software Eng. 1996. No. 3.

141. *Alur R., Henzinger T., Pei-Hsin Ho.* Automatic symbolic verification of embedded systems // IEEE Trans. on Software Eng. 1996. No. 3.

142. *Heimdahl Mats P. E., Leveson N. G.* Completeness and consistency in hierarchical state-based requirements // IEEE Trans. on Software Eng. 1996. No. 6.

143. *Ardis M. A. et al.* A framework for evaluaing specification methods for reactive systems experience report // IEEE Trans. on Software Eng. 1996. No. 6.

144. *Corbett J. C.* Timing analysis of Ada tasking programs // IEEE Trans. on Software Eng. 1996. No. 7.

145. *Zave P., Jackson M.* Where do operations come from? A multiparadigm specification technique // IEEE Trans. on Software Eng. 1996. No. 7.

146. *Coen-Porisini A., Ghezzi C., Kemmerer R.* Specification of real-time systems using ASTRAL // IEEE Trans. on Software Eng. 1997. No. 9.

147. *Avrynin G. S., Corbett J. C., Dillon L. K.* Analyzing partially implemented real-time systems // IEEE Trans. on Software Eng. 1998. No. 8.

148. *Птипьер К.* Синхронный C++ для интерактивных приложений // Открытые системы. 1999. № 3.

149. *Любченко В. С.* О бильярде с Microsoft Visual C++ 5.0 // Мир ПК. 1998. № 1.

150. *Любченко В. С.* Задача Майхилла для Microsoft Visual C++ 5.0 (о синхронизации процессов в среде Windows) // Мир ПК. 2000. № 2.

151. Секреты программирования игр / А. Ла Мот, Д. Ратклифф, М. Семинаторе и др. СПб.: Питер, 1995.

152. *Шоломов Л. А.* Основы теории дискретных логических и вычислительных устройств. М.: Наука, 1980.

153. *Кузнецов О. П.* Неклассические парадигмы в искусственном интеллекте // Изв. РАН. Теория и системы управления. 1995. № 3.

154. *Кук Д., Урбан Д., Хамилтон С.* Unix и не только. Интервью с Кеном Томпсоном // Открытые системы. 1999. № 4.

155. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления // Промышленные АСУ и контроллеры. 1999. № 9.

156. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления // Международная конф. по проблемам управления. М.: Ин-т проблем управления. 1999. Т. 3.

157. *Гудман С., Хидетниеми С.* Введение в разработку и анализ алгоритмов. М.: Мир, 1981.

158. *Брукс Ф.* Мифический человеко-месяц или как создаются программные системы. СПб.: Символ, 2000.

159. *Липаев В. В.* Документирование и управление конфигурацией программных средств. Методы и стандарты. М.: Синтег, 1998.

160. *Непомнящий В. А., Рякин О. М.* Прикладные методы верификации программ. М.: Радио и связь, 1988.

161. *Woodcock J., Davies J.* Using Z-specification. Refinement and proof. Oxford: Oxford University Press, 1995.

162. *Варшавский В. И.* Коллективное поведение автоматов. М.: Наука, 1973.

163. Круглый стол "Парадигмы искусственного интеллекта" // Новости искусственного интеллекта. 1998. № 3.

164. *Воас Д.* Качество ПО: восемь мифов // Открытые системы. 1999. № 9–10.

165. *Антонов А. П., Мелехин В. Ф., Филиппов А. С.* Обзор элементной базы фирмы ALTERA. СПб.: ЭФО, 1997.

166. *Wirth N.* Digital circuit design. NY: Springer-Verlag, 1995.

167. *Wirth N.* Hardware compilation: translating programs into circuits // Computer. 1998. June. (Открытые системы. 1998. № 4–5).

168. *Армстронг Д.* Моделирование цифровых систем на языке VHDL. М.: Мир, 1992.

169. MAX + PLUS II. AHDL. Version 6.0. CA: Altera, 1995.

170. Xilinx Foundation. M.1.5. CA: Xilinx, 1997.

171. *Clare C. R.* Designing logic systems using state machines. NY: McGraw-Hill, 1973.

172. A framework for hardware-software Co-Design of embedded systems // Technical report. Comp. Science Department. Univ. of California. Berkley. 1995.

173. Лекции лауреатов премии Тьюринга за первые двадцать лет 1966–1985. М.: Мир, 1993.

174. *Дейкстра Э.* Взаимодействие последовательных процессов // Языки программирования. М.: Мир, 1972.

175. *Дейтел Х. М., Дейтел П. Д.* Как программировать на С++. М.: Бином, 1999.

176. TSX Nano. PL7-07 language self-instruction manual. Groupe Schneider. 1997.

177. *Поттосин И. В.* О критериях добротности программ // Системная информатика. Вып. 6. Новосибирск: Наука, 1998.

178. *Герр Р.* Отладка человечества // PC Magazine / Russian Edition. 2000. № 5.

179. *Черняк Л.* XML, взгляд со стороны // Открытые системы. 2000. № 4.

180. *Астров В. В., Василенко В. С., Тотьменинов Л. В.* Вопросы создания интегрированных систем управления ядерными энергетическими установками // Системы управления и обработки информации. СПб.: НПО "Аврора". 2000. Вып. 1.

181. *Боггс У., Боггс М.* UML и Rational Rose. М.: Лори, 2000.

182. *Романовский И. В.* Дискретный анализ. СПб.: Невский диалект, 2000.

183. *Тьюринг А.* Может ли машина мыслить? Саратов: Колледж, 1999.

184. *Фон Нейман Дж.* Общая и логическая теория автоматов / Тьюринг А. Может ли машина мыслить? Саратов: Колледж, 1999.

185. *Дейл Н., Уимз Ч., Хедингтон М.* Программирование на С++. М.: ДМК.

Статья представлена к публикации членом редколлегии О. П. Кузнецовым.

Поступила в редакцию 13.06.99