Глава 19

SWITCH-технология. Функциональное программирование без программистов

Результаты, изложенные в настоящей работе, могут использоваться при различных подходах к программной реализации алгоритмов логического управления.

По мнению автора, с появлением современных промышленных компьютеров наибольший интерес представляет случай программной реализации алгоритмов указанного класса в базисе языков программирования высокого уровня.

Ниже, обобщая изложенное, формулируются основные положения подхода, предлагаемого автором и названного им SWITCH-технологией. Подход позволяет проводить функциональное программирование без программистов.

Предложенная технология может быть названа также STATE-технология или более точно AUTOMATON-технология.

Завершив разработку технологии, автором был выполнен дополнительный анализ литературы с целью более детального рассмотрения работ, содержащих идеи, наиболее близкие к предлагаемым.

19.1. Предшествующие результаты

В [61] предложен язык «Ярус» для описания алгоритмов логического управления, разработанный под руководством О. П. Кузнецова в Институте проблем управления (ИПУ), г. Москва, Россия. В дальнейшем применительно к программной реализации алгоритмов был разработан язык «Ярус-2» [155].

При применении языка «Ярус» отдельные части алгоритма, названные пунктами, описываются с помощью графов переключений, в которых на дугах в отличие от графов переходов для автоматов Мили записываются не все выходные переменные или их инверсии, а лишь те из них, значения которых изменяются на рассматриваемом переходе, в то время как значения остальных переменных умалчиваются [120].

Это позволяет в ряде случаев реализовать заданный алгоритм с помощью графа переключений с меньшим числом вершин, чем в соответствующем графе переходов. Поэтому в отличие от термина «состоя-

ние», используемого в графах переходов, для графов переключений был введен термин «ситуация» (который по смыслу отличается от аналогичного термина, используемого в [281]).

Связь между графами переключений в системе взаимосвязанных графов осуществляется, как и в предлагаемом подходе, с помощью значений многозначных переменных, названных ситуационными. Обеспечена возможность реализации параллельных процессов.

Наряду с достоинствами, ряд из которых перечислен выше, язык «Ярус» обладает и недостатками, а именно:

— разработка собственного синтаксиса, связанная с ранним появлением языка (1971—1972 гг.), что ограничило широкое его применение;

— термин «ситуация» менее понятен, чем термин «состояние»;

— размещение значений выходных переменных на дугах менее понятно, чем их расположение в вершинах;

— указание на дугах только значений изменяющихся переменных затрудняет чтение спецификации и внесение изменений в нее из-за введения глубокой предыстории в процесс алгоритмизации.

В работах В. В. Руднева (ИПУ) [63, 158—164], выполненных после разработки языка «Ярус», обоснован переход от модели конечного автомата, задаваемой графом переходов, к системе взаимосвязанных графов переходов и показаны широкие функциональные возможности этой модели, в том числе и при реализации параллельных процессов. Для связи графов используются двоичные переменные, названные переменными связи. Рассмотрен вопрос о декомпозиции графа на систему взаимосвязанных графов.

Эти работы носят теоретический характер и были недостаточно ориентированы на практическое применение и программную реализацию.

Язык графов операций, разработанный С. А. Юдицким (ИПУ), и язык «Графсет», базирующийся на сетях Петри, рассмотрены в гл. 11, 12 соответственно.

В [175] было предложено использовать ГП при программной реализации алгоритмов логического управления технологическими процессами. Однако изложенный в этой работе подход был недостаточно «изящным», что не позволило ему найти широкое практическое применение.

В работах по программированию для системы «AutoCAD» [166] и для системы «Selma-2» [167] было предложено для программной реализации автоматов применять конструкцию switch.

Однако вопросы программной реализации взаимодействующих автоматов, их композиции, декомпозиции, вызываемости, вложенности и ряд других в работе [166] не рассматривались.

Сравнительно новой является концепция создания программ, управляемых событиями [280]. При этом под «событием» понимается то, что происходит в реальном времени и вызывает те или иные действия. Основой программирования по событиям является определение ответных действий, соответствующих данному событию [318].

На программном уровне сопоставление событий и действий в [280] осуществляется с помощью конструкции case. . .of языка «Паскаль», которая аналогична конструкции switch языка СИ.

В отличие от технологии, предлагаемой в настоящей работе, в которой основными являются понятия «состояние» и «последовательностный автомат», в [280] эти понятия не используются, а конструкция `case... of` применяется в основном для комбинационного выбора действий по событиям. Более подробно этот вопрос рассмотрен в разд. 19.6 и Приложении 11.

При этом отметим, что если использовать подход, предлагаемый в настоящей работе, то понятие «событие» целесообразно связывать с изменениями значений только входных, временных и выходных переменных, в то время как для внутренних переменных должно применяться понятие «состояние». При этом для взаимодействия параллельных процессов, протекающих в алгоритмах, каждый из которых может быть описан с помощью графа переходов последовательностного автомата, в настоящей работе в качестве основы предлагается использовать значения многозначных переменных, кодирующих состояния этих автоматов.

19.2. Основные положения

Предлагаемый в настоящей работе подход, впервые сформулированный в [167, 214, 275], базируется на следующих основных положениях.

Проектирование должно начинаться с построения схемы связей «ИИ—У А—СПИ—ИМ», которая в дальнейшем преобразуется в схему связей «ИИ—А—ФЭЗ—СПИ—ИМ».

При необходимости автомат может быть декомпозирован на систему взаимосвязанных автоматов (СВА), взаимодействующих между собой, например по принципу вызываемости или принципу вложенности.

Этап архитектурного (системного) проектирования в этом случае завершается построением схемы связей «ИИ—СВА—ФЭЗ—СПИ—ИМ».

Синхронность программной реализации алгоритмов логического управления делает автоматные модели в отличие от асинхронной аппаратной реализации, применяемой до настоящего времени в системах управления, адекватно выполнимыми, что обеспечивает возможность широкого применения таких моделей на практике в этом случае.

Основным для автоматов является понятие «состояние», являющееся абстрактным понятием, отражающим некоторую стадию реализации алгоритма. Для предания состоянию физического смысла в большинстве случаев предлагается связывать с ним одну из комбинаций значений выходных переменных и состояние объекта управления (разд. 4.4.9).

При этом число состояний в автомате должно быть не меньше числа различных комбинаций выходных переменных, которые он должен формировать. В простейших автоматах каждой комбинации выходных значений соответствует одно состояние, а для более сложных автоматов одна и та же комбинация может формироваться в разных состояниях. Эти состояния могут различаться не введением дополнительных разрядов (двоичных внутренних переменных) в вектор выходных переменных, а присвоением состояниям десятичных номеров — многозначных кодов (одной многозначной внутренней переменной).

При этом отсутствует необходимость устанавливать и сбрасывать большое число двоичных внутренних переменных, так как для каждого графа переходов они могут быть заменены лишь одной многозначной переменной, предыдущее значение которой сбрасывается автоматически при присвоении ей нового значения. Эта переменная принимает число значений, равное числу состояний автомата, что резко повышает наглядность описания поведения автоматов и существенно отличает этот подход от используемых в логическом проектировании схем.

Функционирование автомата происходит следующим образом: определяется состояние, в котором он находится; определяются значения входных переменных; вычисляется следующее состояние автомата. Место формирования комбинации значений выходных переменных в этой последовательности действий определяется типом автомата, а наличие или отсутствие задержки на программный цикл их изменения относительно изменения значения переменной состояния — родом автомата.

При этом в качестве основной структурной модели автомата в системах логического управления предлагается использовать автомат Мура второго рода, в котором значения выходных переменных зависят лишь от состояния автомата, но не зависят от входных воздействий, а указанная задержка отсутствует.

Важная особенность предлагаемого подхода состоит в том, что в большинстве случаев в каждом состоянии указываются значения всех выходных переменных, а применение умолчаний запрещается. При этом резко упрощается внесение изменений, так как значения выходных переменных не зависят от предыстории, т. е. значений выходных переменных в других состояниях. Запрещается также использование флагов (гл. 9).

Поведение (динамику) автоматов будем задавать (описывать) с помощью графов переходов, состоящих из двух помеченных компонент — вершин и дуг.

Вершины соответствуют состояниям автомата Мура и помечаются дробью, числителем в которой является номер состояния, а знаменателем — комбинация значений всех выходных переменных, формируемых в этом состоянии.

Каждая дуга в графе переходов отмечается булевой формулой, при равенстве единице которой осуществляется переход из вершины, соответствующей началу дуги, в вершину, соответствующую концу дуги. При этом начало и конец дуги могут совпадать и тогда им будет соответствовать одна вершина. В этом случае говорят, что имеет место петля, обеспечивающая сохранение состояния автомата до тех пор, пока выполняется условие, ее помечающее, а вершину с петлей называют устойчивой.

Каждая вершина без петли является неустойчивой — в ней автомат находится только один программный цикл. В общем случае и в устойчивой вершине автомат также может «находиться» только один программный цикл.

Символическое обозначение памяти в виде вершины графа переходов существенно более наглядно по сравнению с триггером в функциональной схеме, для понимания работы которого необходимо знать правила его функционирования.

Булева формула, помечающая каждую дугу, зависит не от всех входных переменных автомата, а только от тех из них, которые обеспечивают рассматриваемый переход. Поэтому если в некоторой вершине изменить значение переменной, не указанной ни на одной из исходящих из этой вершины дуг, то автомат своего состояния не изменит.

Граф переходов без флагов и умолчаний описывает поведение автомата (динамику его функционирования) в терминах состояний. Граф переходов, построенный определенным в настоящей работе образом, достаточно просто читается, во-первых, потому что его дуги, исходящие из каждой вершины, помечены не всеми переменными, а лишь теми из них, которые обеспечивают переходы в соседние вершины, во-вторых, по графу переходов явно видно, какие значения и в каком порядке должны принимать какие переменные для обеспечения переходов из рассматриваемой вершины в любую из заданных, и в-третьих, потому что в нем отсутствуют флаги, а в его вершинах указаны значения всех выходных переменных, исключая тем самым их зависимость от предыстории.

Это позволяет читать граф переходов в отличие, например, от функциональной схемы, по которой приходится считать, так как она не содержит в явном виде значений выходных переменных. При этом таблица проверки графа переходов автомата с памятью должна состоять не из двух столбцов: «Вход», «Выход», что достаточно лишь для проверки автоматов без памяти, а из четырех столбцов, в первых двух из которых указываются номер текущего состояния и булева формула, обеспечивающая переход в следующее состояние, номер которого записывается в третьем столбце. Четвертый столбец содержит значения выходных переменных в новом состоянии. Каждая строка этой таблицы в свою очередь может быть преобразована в совокупность строк, число которых равно числу наборов входных переменных. Таблица проверки для автоматов Мура может быть также разбита и на две таблицы, первая из которых содержит первые три указанных выше столбца, а вторая — два: «Состояние», «Выход».

Граф переходов не должен содержать генерирующих контуров, а для каждой его вершины должны быть обеспечены полнота и непротиворечивость. Полнота понимается в том смысле, что для любого входного набора переменных, помечающих исходящие из этой вершины дуги, определен переход в одну из вершин графа. В математической форме условие полноты выражается так: дизъюнкция пометок всех исходящих из вершины дуг должна быть тождественно равна единице.

Непротиворечивость понимается в том смысле, что не должно существовать ни одного входного набора, при котором выполняются условия, обеспечивающие возможность перехода более чем в одну вершину графа переходов. В математической форме это выражается следующим образом: конъюнкция пометок двух любых исходящих из вершины дуг должна быть тождественно равна нулю.

Обеспечение полноты и непротиворечивости делает язык графов переходов анкетным языком и приводит Разработчика к необходимости ставить перед Заказчиком и (или) Технологом вопросы для обеспечения указанных свойств [179].

Обеспечение непротиворечивости является необходимым, а полнота в явном виде может требоваться не для всех алгоритмических и программных моделей.

Достижение полноты в указанном виде в ряде случаев является весьма трудоемким, но бывает необходимым, например для получения СБФ, эквивалентной графу переходов.

При реализации графа переходов с помощью конструкции switch ее структура позволяет обеспечить первое из указанных свойств только расстановкой приоритетов для всех помеченных дуг, исходящих из каждой вершины графа, а второе — без пометки петель. Это существенно упрощает переход от спецификации к программной реализации алгоритмов при применении языков высокого уровня, содержащих в своем составе указанную конструкцию или ее аналог.

При использовании графа переходов по номеру его вершины (многозначному коду состояния) можно определить, где находится автомат в данный момент времени в пространстве состояний, какие значения при этом принимают все его выходные переменные и какие входные переменные и при каких значениях обеспечивают переход в следующее состояние.

Без применения понятия «состояние» определить «местонахождение» в указанном пространстве автомата с памятью, формирующего в том числе и одинаковые комбинации значений выходных переменных, невозможно.

Из изложенного следует, что графы переходов целесообразно применять в качестве языка спецификации.

Применение этого языка совместно с графами достижимых маркировок позволяет проводить доскональный анализ алгоритмов до их программирования.

Для подтверждения выводов, сформулированных выше, изложим историю, происшедшую с автором, в форме диалога между ним и Разработчиком, которая явилась импульсом для написания настоящей книги.

Разработчик:

— Как записать формулу триггера?

Автор:

— Какого типа триггер?

(Легкое замешательство Разработчика).

Автор:

— Для какой цели эта формула требуется?

Разработч ик:

— На ПЭВМ необходимо реализовать алгоритм управления клапаном, который я описал функциональной схемой, а затем выполнить тестирование этого алгоритма.

Автор:

— Каким образом Вы собираетесь проводить тестирование?

Разработчик:

— Подам входные воздействия с клавиатуры ПЭВМ и буду наблюдать на ее дисплее выходные реакции.

Автор:

— Сколько в схеме входов и триггеров?

Разработчик:

— Шесть и четыре.

Автор:

— Вы будете в каждом из теоретически возможных 16 состояний подавать все 64 входных набора и анализировать все выходные реакции?

Разработчик:

— Нет. Подаваться будут только рабочие входные наборы.

(Вопрос о состояниях остался вообще незамеченным).

Автор:

— Что Вы при этом проверите?

(Замешательство Разработчика).

Разработчик:

— А что предлагаете Вы?

Автор предложил по функциональной схеме, определив тип используемых триггеров, записать систему булевых формул и формально построить по ней граф переходов.

После этого Автор предложил Разработчику выполнить анализ этого графа с целью определения, соответствует ли поведение построенной схемы желаемому.

Проанализировав каждый переход графа, Разработчик обнаружил, что один из них некорректен. (Тестированием это скорее всего вообще не было бы обнаружено, так как указанный переход осуществлялся при нерабочем входном наборе).

Разработчик:

— Я пошел корректировать схему.

Автор:

— Каким образом Вы собираетесь это сделать? Вновь эвристически? А мне снова строить граф переходов по новой схеме, которая может содержать уже другие ошибки?

Разработчик (раздраженно):

— А что Вы предлагаете?

Автор предложил вместо исправления схемы внести изменения в граф переходов с целью обеспечения желаемого его поведения, что и было сделано Разработчиком.

Автор:

— Вы действительно для программной реализации алгоритма хотите в качестве спецификации применять функциональную схему или Вас устроит любое другое описание алгоритма, по которому построение корректной программы выполняется проще?

Немая сцена.

После этого автор предложил использовать в качестве спецификации откорректированный граф переходов, выполнил многозначное кодирование вершин графа, формально реализовал его одной конструкцией switch языка СИ и осуществил на ПЭВМ сертификацию построенной программы, применяя в качестве теста этот граф переходов.

После этого у Разработчика навсегда отпало желание использовать эвристически построенные функциональные схемы (как, впрочем, и любые другие) для спецификации алгоритмов логического управления при их программной реализации.

При использовании других языков, отличных от графов переходов, для спецификации указанных алгоритмов возникают проблемы с тес-

тированием и внесением изменений, описанные в приведенной выше истории.

Графы переходов в качестве языка спецификаций могут применяться не только для описания автоматов, реализующих алгоритмы логического управления, но и при создании моделей объектов, управляемых этими автоматами, что позволяет с единых позиций описывать комплекс «УА—ОУ».

В качестве языка реализации графов переходов может быть выбран практически любой язык программирования, например язык функциональных схем, как это имеет место во многих ПЛК. Среди алгоритмических языков высокого уровня наиболее целесообразно выбрать тот из них, который содержит конструкцию, позволяющую изоморфно отображать графы переходов в текстах программ.

В случае существования такой конструкции появляется возможность проводить проверку корректности перехода от спецификации к программе не в динамике (тестированием), а в статике — сверкой текста программы с графом переходов без флагов и умолчаний, по которому она строилась и который должен быть согласован с Заказчиком. При таком подходе наличие ПЭВМ мало что добавляет.

Как отмечалось выше, такой конструкцией является, например, конструкция `switch`, входящая в состав языка СИ. Существуют различные варианты реализации графа переходов с помощью конструкции `switch`. Рассмотрим один из них применительно к автоматам Мура первого рода. При этом графу переходов присваивается переменная *Y,* число значений которой равно числу вершин *s* в графе. Число меток `case`, входящих в состав конструкции `switch`, в этом случае равно *s*. В метке `case`, соответствующей *i*-й вершине графа, в первой строке перечисляются выходные переменные и принимаемые ими значения, а в последней строке указывается оператор `break`. Между первой и последней строками размещаются операторы

$$\texttt{if}\ (f_{ij})\quad Y = j,$$

где f_{ij} — булева формула, помечающая дугу между вершинами с номерами i и j $(i \neq j)$

Число операторов if в метке `case` равно числу исходящих из рассматриваемой вершины дуг (без учета петли). При наличии в вершине петли она реализуется указанным выше оператором `break`, который обеспечивает также и выход из оператора `switch` после реализации не более чем одного перехода в ГП.

При такой организации программы обеспечение непротиворечивости и полноты может быть выполнено только расстановкой приоритетов. При этом формула с наибольшим приоритетом должна размещаться в последнем операторе if.

Предлагаемая стандартная реализация алгоритмов логического управления, базирующаяся на использовании конструкции `switch`, обеспечивает линейную зависимость сложности программы от параметров исходного описания (спецификации), что чрезвычайно важно для обеспечения возможности широкого практического применения предлагаемого подхода.

При этом отметим, что определение классов задач и методов их решения, для которых могут быть предложены стандартные реализации

и найдены линейные оценки сложности, является чрезвычайно важным и для других областей логического проектирования [39, 157].

Если заданный алгоритм декомпозируется, то он реализуется не одним графом, а взаимосвязанной системой графов переходов.

Взаимосвязь графов в системе может осуществляться, например, с помощью входных и выходных переменных, а также переменных состояний. При этом если входные и выходные переменные двоичны, то переменные, кодирующие состояния, многозначны. Последние применяются для связи графов переходов (операторов switch) в форме условия $Y_i == j$, вырабатывающего значение, равное единице, если номер Y_i равен j, и ноль — во всех остальных случаях.

Выходные переменные для связи графов переходов рекомендуется применять лишь в тех случаях, когда они позволяют упростить пометку дуг и это упрощение необходимо. Пусть, например, имеется система, состоящая из двух взаимосвязанных графов, и требуется осуществить некоторый переход во втором графе, если первый граф находится в первом или во втором состоянии. Это условие записывается следующим образом: $(Y_1 == 1) \vee (Y_1 == 2)$. Однако в случае, если только в этих состояниях i-я выходная переменная z_i равна единице, на соответствующей дуге второго графа достаточно записать z_i.

Использование многозначных переменных, кодирующих состояния, для обеспечения связи графов переходов в системе принципиально отличает предлагаемый подход от изложенного в [160]. Это минимизирует число объявляемых переменных и позволяет обеспечить взаимосвязь графов в наглядной форме.

Программа, созданная с помощью предлагаемого подхода, обладает свойством потенциальной наблюдаемости с помощью минимального числа внутренних переменных. Для перехода от потенциальной наблюдаемости к реальной разработана программная оболочка, обеспечивающая возможность для программы, записанной на языке СИ, изменять значения входных переменных и наблюдать на дисплее «вычисляемые» программой значения выходных и временных переменных.

Основное достоинство оболочки, отличающее ее от известных, состоит в том, что она позволяет также наблюдать многозначный номер состояния, в котором каждый автомат находится в данный момент времени, без необходимости принудительного вызова Оператором соответствующих переменных. Изменения номеров состояний под воздействием изменений значений входных переменных позволяют прослеживать траекторию «движения» автомата в пространстве состояний, определяемом его графом переходов.

Организация взаимосвязей между графами с помощью переменных состояний позволяет на языке графов переходов в наглядной форме строить головной и вызываемые графы, позволяющие описывать в том числе и параллельно-последовательные процессы. При этом взаимодействие графов в системе осуществляется по принципу «запрос—ответ-сброс» или «запрос—ответ».

Рассмотренные разновидности взаимосвязи между графами переходов являются связями по данным, а связи между ними по управлению обычно в явном виде (дугами) не изображаются, а реализуются с помощью

последовательного их расположения. (Исключение из этого правила рассмотрено в разд. 8.7.2).

На программном уровне последовательному расположению графов переходов соответствует управляющая структура «последовательность» [103], т. е. последовательное расположение конструкций `switch`.

Программа, построенная указанным образом, функционирует так, что за один ее проход для каждого графа переходов либо сохраняется текущее состояние, либо реализуется только один переход из рассматриваемого состояния в одно из смежных.

Получающиеся при этом тексты программ однородны и структурированы, а по ним просто и однозначно могут быть восстановлены спецификации — системы графов переходов, по которым эти программы строились. Получаемые программы не только наблюдаемы, но и управляемы, так как входные, выходные и внутренние переменные в них легко идентифицируются, а в тексты программ легко и корректно могут быть внесены изменения.

В рамках предполагаемого подхода удается осуществлять реализацию алгоритмов по принципу сверху вниз, при использовании которого первоначально строится головной граф, а во всех вызываемых графах их тела заменяются вершинами-заглушками. Это позволяет производить отладку связей между графами переходов системы-прототипа с помощью ПЭВМ на начальной стадии проектирования. После обеспечения правильного взаимодействия системы графов переходов можно производить разработку графов, замещающих вершины-заглушки, в том числе и параллельно несколькими специалистами.

В качестве тестов для демонстрации правильности функционирования программы, реализующей систему графов переходов, можно построить и использовать граф проверки, соответствующий желаемому поведению системы и позволяющий подтвердить или опровергнуть такое поведение. Если система графов переходов обладает дополнительными функциональными возможностями по сравнению с желаемыми, то граф проверки этого не обнаруживает, что весьма опасно. Все функциональные возможности системы графов переходов или одного ГП с флагами и (или) умолчаниями могут быть определены с помощью графа достижимых маркировок, который в случае его правильности и является полным проверяющим тестом.

Если параллелизм по состояниям отсутствует, то сложность описания системы графов переходов совпадает со сложностью ее поведения, определяемого графом достижимых маркировок. При наличии этой разновидности параллелизма сложность графа достижимых маркировок превышает сложность описания системы графов переходов.

Кроме описанного варианта взаимодействия графов переходов допустимо также и их вложение, что программно реализуется с помощью вложенных конструкций `switch`.

Если в различные вершины некоторого графа переходов «вкладываются» однотипные графы переходов следующего уровня иерархии, то последние в спецификации могут быть изображены однократно, а для программной реализации этой системы может быть использован механизм процедур (разд. 12.4).

Удобство использования конструкции `switch` или ее аналогов в предлагаемой технологии позволяет применять широко распространенные языки программирования высокого уровня, а не разрабатывать новый язык. В этой технологии синтаксис языка спецификаций содержит простейший набор из двух компонент — помеченных вершин и дуг, а синтаксис языка программирования — в качестве основной содержит единственную конструкцию `switch`.

При необходимости оптимизация программы по памяти и (или) быстродействию первоначально должна производиться за счет раздельной минимизации булевых формул, входящих в операторы `if` конструкции `switch`. Если после этого заданные требования остаются невыполненными, то в операторах `if` булевы формулы заменяются переменными, а сами формулы выносятся из операторов `switch`, образуя систему, формулы которой минимизируются совместно. Если и после совместной минимизации этой системы, проводимой без разрушения части программы, обеспечивающей переходы и реализуемой конструкциями `switch`, требования по указанным показателям не выполняются, то может быть произведена поэтапная замена и отдельных операторов `switch` на менее наглядное, но более компактное описание, например с помощью СБФ.

В рамках предлагаемого подхода при наличии схемы связей «ИИ—СВА—ФЭЗ—СПИ—ИМ» и системы взаимосвязанных графов переходов можно практически полностью отказаться от применения словесных описаний алгоритмов логического управления в технической и программной документации, включив в нее также правила чтения графов переходов и графов достижимых маркировок.

Используемые в настоящее время словесные описания отображают лишь некоторое ядро каждого алгоритма, а не весь алгоритм в целом, так как в противном случае такое описание было бы чрезвычайно громоздким и нечитаемым. Словесное описание ядра может быть сохранено также и при наличии в составе документации графов переходов, однако лишь в качестве комментария. Более того, изоморфизм между графом переходов и конструкцией `switch` позволяет в принципе отказаться и от введения системы ГП в документацию, заменив ее при необходимости системой взаимосвязанных конструкций `switch`. При этом осуществляется переход от алгоритмической модели к функциональной программе на языке высокого уровня, практически не теряя наглядности описания по сравнению с системой ГП. Последняя всегда может быть просто и однозначно восстановлена по тексту программы.

Изложенный подход может быть использован не только для программной реализации алгоритмов логического управления, но и для других классов алгоритмов. В рамках предлагаемой технологии могут применяться и другие модели автоматов, рассмотренные в настоящей работе, отличные от автоматов Мура.

В заключение раздела отметим, что предлагаемая SWITCH-технология может рассматриваться как одно из направлений широко развиваемой в настоящее время CASE-технологии (Computer Aided Software Engineering) [216], «которой практически нет альтернативы, так как без специальной методики и инструментария составить техническое задание на крупную систему, адекватно описывающее все нюансы ее поведения,

практически невозможно. Кроме того, Заказчик и Разработчик как специалисты разного профиля часто говорят на разных языках, понимая под одним и тем же пунктом задания порой совершенно разные вещи.

Преодолеть эти разногласия позволяет CASE-технология, основная идея которой состоит в том, чтобы описывать будущую систему в форме неких конструкций, одинаково понятных Заказчику и Разработчику.

При этом Заказчик получает возможность понять, что можно ждать от системы, а Разработчик, в свою очередь, может точно определить требования Заказчика, быстро создать прототип будущей системы и объяснить Заказчику, как она будет работать. Он может проверить на непротиворечивость и полноту исходные данные и объяснить Заказчику, каких данных и почему не хватает. Предлагаемая технология позволяет эффективно использовать персонал средней квалификации при реализации системы» [217].

Изложенное особенно важно, если учитывать один из законов Мэрфи, по которому «если что-то кажется простым, то обычно является сложным, а если кажется сложным, то может оказаться и вовсе не выполнимым» [275].

19.3. Стандарт IEC 1131 и SWITCH-технология

В системах логического управления на нижнем уровне приборной структуры используются три типа вычислительных устройств: промышленные компьютеры, контроллеры и программируемые логические контроллеры, для каждого из которых применяются соответствующие языки программирования.

Промышленные компьютеры (ПК) программируются обычно с помощью алгоритмических языков высокого уровня, например СИ.

Контроллеры реализуются как на базе микропроцессоров (например, семейств i286, i386), так и на базе микроконтроллеров (например, семейств MCS-51, MCS-96, MCS-151, MCS-251). В первом случае программирование выполняется либо на соответствующем ассемблере, либо, например, на языке СИ, а во втором — на ассемблерах, более приспособленных для реализации функций управления, а также на языках высокого уровня таких, например, как COMBASIC, PL/M, СИ.

Программируемые логические контроллеры (ПЛК) могут быть реализованы как на базе микропроцессоров, так и микроконтроллеров. Существуют ПЛК, например TSX 107-40 фирмы «Телемеханик», процессорный блок которых содержит цифровой микропроцессор i80386, коммуникационный микропроцессор i80C52 и булевский микропроцессор для реализации последовательностных функций.

Возникает вопрос, почему при столь высоком уровне развития ПК до сих пор при создании систем управления широко применяются ПЛК? Один из ответов на этот вопрос состоит в следующем: «Заказчики избегают непосредственного управления от ПК ответственными производственными и технологическими объектами из-за ненадежности современных операционных систем (ОС). Зависание ОС, случающееся в ПК достаточно часто, неминуемо приводит к выходу из строя всей системы

и к остановке производственного или технологического цикла, что во многих случаях совершенно недопустимо. Использование на нижнем уровне надежных ПЛК, управляемых от ПК, практически исключает прерывание цикла работы на этом уровне, в том числе и при сбое и отказе в работе ОС компьютера. На практике прерывание цикла нормальной работы ПЛК может произойти лишь при отключении питания» [300]. Этот вывод подтверждается и практическим опытом автора.

Другая особенность ПЛК состоит в том, что они имеют принципиально другие языки программирования, еще более приспособленные для целей управления, чем языки контроллеров, указанные выше, так как учитывают не только опыт программистов, но и разработчиков схем.

Эти языки делятся на три класса — текстовые, графические и смешанные.

В 1993 году Международная электротехническая комиссия выпустила стандарт IEC 1131, посвященный программируемым логическим контроллерам, в третьей части которого описываются языки их программирования [260].

В соответствии с этим стандартом к текстовым языкам принадлежат:

— языки инструкций («Instruction List» — IL), в которых в отличие от ассемблеров мнемоника лучше приспособлена для написания и чтения управляющих программ;

— структурный язык («Structured Text» — ST), являющийся простейшим языком высокого уровня.

При этом отметим, что среди рекомендованных стандартом операторов имеется оператор `case` (аналог конструкции `switch`), и поэтому для ПЛК тех фирм, которые включили его в систему «команд» (например, [341]), SWITCH-технология может использоваться непосредственно.

К графическим языкам принадлежат:

— язык лестничных схем («Ladder Diagram» — LD), базовые символы которого предназначены для описания релейно-контактных схем;

— язык функциональных схем («Function Block Diagram» — FBD).

При этом отметим, что среди рекомендованных стандартом блоков имеется цифровой мультиплексор(аналог конструкции `switch`), и поэтому для ПЛК тех фирм, которые включили его в библиотеку блоков (например, [60]), SWITCH-технология может применяться практически непосредственно.

К языкам смешанного типа относится язык последовательных функциональных диаграмм («Sequential Function Chart» — SFC). При использовании этого языка процесс представляется в виде шагов (Steps), связанных между собой переходами (Transition), каждый из которых помечается условием (Condition). С каждым шагом диаграммы сопоставляется (как это имеет место только в автоматах Мура) действие (Action). При этом для описания условий, и в особенности действий, могут применяться указанные выше текстовые и графические языки, что затрудняет понимание диаграмм, тем более что это делается на других «экранах». Несмотря на название — «последовательные», они могут использоваться для описания в одной компоненте параллельно-последовательных процессов. Если параллельные процессы являются взаимосвязанными, то, несмотря на то что с помощью рассмотренного языка они могут быть заданы весьма компактно, полный анализ их совместного поведения весьма сложен, так

как одновременно приходится «следить» за несколькими «активными» вершинами. Это требует построения графа достижимых маркировок, эквивалентного графу переходов с большим числом вершин, в котором в каждый момент времени только одна вершина является активной, что и упрощает анализ последнего.

В этом смысле язык SFC с параллельными взаимосвязанными процессами напоминает мостиковые контактные схемы, в то время как графы переходов без флагов и умолчаний — параллельно-последовательные контактные схемы. Несмотря на то что отдельные алгоритмы описываются более компактно мостиковыми схемами, на практике в основном применяются параллельно-последовательные, для которых имеется изоморфизм с формулами булевой алгебры.

Поэтому если граф достижимых маркировок использовать в качестве управляющего, то его структура будет совпадать с описанием его поведения, правда, за счет, возможно, весьма значительного увеличения числа вершин по сравнению с SFC-диаграммой. Одной из реализаций языка SFC является язык «Графсет» [59].

Основной недостаток рассматриваемого стандарта состоит в том, что, описывая синтаксис языков и их операторов, он содержит лишь отдельные примеры (окончательные реализации) и не включает методов алгоритмизации и программирования в базисе этих языков. При этом если язык SFC является не только языком программирования, но и весьма удобным языком алгоритмизации, то, как следует из содержания настоящей книги, применение остальных языков программирования, описываемых в рассматриваемом стандарте, для алгоритмизации процессов управления нецелесообразно.

Однако, как показывает опыт общения автора с Практиками, не язык SFC (даже при наличии транслятора с него), а язык инструкций и (или) лестничные и функциональные схемы (как более привычные) используются чаще всего в качестве языков алгоритмизации и одновременно языков программирования при условии, что в состав системы автоматизации программирования применяемого ПЛК входят трансляторы с указанных языков.

На широкое применение языка SFC практически не влияют даже примеры, приводимые в отдельных фирменных документах. Так, в документации фирмы «Омрон» [236] на достаточно сложном примере показано, что описание алгоритма на языке лестничных схем по объему более чем в два раза превышает соответствующее описание на языке SFC, не говоря уже о большей понятности последнего.

В России ситуация усугубляется еще и тем, что имеется только стандарт [245], который, по мнению автора, для рассматриваемого класса систем во многом не адекватен, а стандарт, аналогичный стандарту IEC 1131-3, отсутствует, что резко затрудняет внедрение предлагаемой технологии для систем, применяемых Заказчиком. При этом отметим, что фирму «Сименс» не смутило, что диаграммы состояний (графы переходов) не являются языком, рекомендованным стандартом IEC 1131-3, хотя приверженность этому стандарту фирма в своей документации [305] демонстрирует явно.

Эта ситуация может измениться не только с появлением настоящей книги и соответствующих стандартов, а в основном в связи с разработкой

и широким распространением CASE-продуктов для управляющих вычислительных устройств, предназначенных для решения задач рассматриваемого класса.

Так, программный продукт «ISaGRAF» [261], реализующий стандарт ШС 1131-3, в качестве основного использует язык SFC (стандарт IEC 848). Естественно, что такой же подход сохранен и при «привязке» программ ISaGRAF к платформе «MVME162» [262].

Указанный стандарт реализован также, например, в программном продукте «ET-PDS» фирмы «Toshiba» (Япония) [266]. Программный продукт «GELLO» фирмы «Event Technology» (США) также поддерживает этот стандарт [326, 329].

Разработка языков программирования в рамках стандарта IEC 1131-3 проводится и в России. Так, например, язык «Микрол+» [231, 323] относится в соответствии с этим стандартом к языкам типа «структурированный текст».

Основные отличия текстовых языков этого типа от универсальных языков состоят в резком упрощении синтаксиса и семантики и во введении в них специальных технологических понятий, реализующих типовые функции контроля и управления.

В языке «Микрол+» существуют два основных типа операторов: условный и безусловные. Условный оператор имеет вид «Если ... Тогда... Иначе ...». Безусловные операторы — это арифметические выражения, оператор перехода на метку (только вперед по тексту программы), вызов процедуры и т. д.

При разработке этого языка было принято решение не применять операторы цикла, поскольку они могут быть реализованы за счет циклического выполнения программы в целом, а их наличие в программе не только приводит к ее потенциальной ненадежности, но делает невозможным любой прогноз относительно гарантированного времени выполнения.

Пользователь может формировать библиотеки из программ, написанных как на языке «Микрол+», так и на языке СИ. Созданная программа транслируется входящим в состав инструментальной среды «MicPlus» компилятором, который создает не исполняемый код, а промежуточное представление — программу на языке СИ. Именно благодаря такому подходу обеспечивается легкий перенос системы программирования на различные целевые платформы, а также открытость и расширяемость. (Отметим, что и в предлагаемой автором SWITCH-технологии также рекомендуется использовать двухуровневую трансляцию, как например это имеет место при следующих преобразованиях: «система взаимосвязанных графов переходов — программа на языке СИ»; «программа на языке СИ — программа на языке ALPro»).

Из изложенного следует, что и при применении языка «Микрол+» остается открытым вопрос о предварительной алгоритмизации и формальном, и изоморфном программировании в его базисе. Решение этого вопроса может быть выполнено на основе SWITCH-технологии.

Сформулируем основные особенности этой технологии.

Для создания «понятных» спецификаций и обеспечения формального и изоморфного перехода от них к программам в базисе различных языков, перечисленных выше, а также для обеспечения максимальной независи-

мости процесса создания программ от типа используемого языка программирования и наличия транслятора с него в настоящей работе предложена SWITCH-технология, в рамках которой разработку программ предлагается проводить в общем случае в четыре стадии:

— архитектурное (системное) проектирование;

— проектирование формальных спецификаций;

— построение и оптимизация алгоритмических моделей, реализующих формальные спецификации;

— программирование.

Предлагаемая технология базируется на автоматном подходе к проектированию программ.

В результате архитектурного проектирования на основе словесного задания и опыта Разработчика эвристически строится схема связей «источники информации—система взаимосвязанных автоматов—функциональные элементы задержки—средства представления информации—исполнительные механизмы».

На второй стадии выбираются:

— графы переходов в качестве языка спецификаций;

— структурная модель каждого автомата;

— вид кодирования состояний каждого автомата.

На этой стадии строится «автоматная» спецификация в общем случае в виде системы взаимосвязанных графов переходов, структура каждого из которых однозначно соответствует выбранным структурной модели и виду кодирования состояний автомата. Обеспечивается формальная корректность каждого графа переходов. Строится граф достижимых маркировок и определяется в результате его анализа правильность «автоматной» спецификации. Строятся корректные спецификации для ФЭЗ и моделей объектов управления.

Разработан механизм, позволяющий без введения дополнительных переменных (отличных от переменных, кодирующих внутренние состояния отдельных автоматов) обеспечить взаимодействие графов переходов, включая иерархию и параллелизм. Показано, что введение дополнительных переменных (флагов) позволяет сократить число вершин в графах переходов за счет ухудшения их читаемости из-за введения зависимости следующего состояния не только от значений входных переменных и состояния в данный момент времени, но и от предшествующих состояний. Рассмотрены также модели, в которых значения выходных переменных в рассматриваемом состоянии зависят от значений этих переменных в предшествующих состояниях. Это в общем случае, позволяя сократить число состояний, осложняет возможность безошибочного внесения изменений.

На третьей стадии осуществляется выбор, построение и оптимизация алгоритмических моделей (см. гл. 4) для реализации «автоматной» спецификации, ФЭЗ и моделей ОУ. При этом учитывается род структурной модели для каждого автомата с памятью.

На четвертой стадии выбирается язык программирования; осуществляется выбор, построение и оптимизация программных моделей для реализации выбранных алгоритмических моделей, соответствующих «автоматной» спецификации и спецификациям ФЭЗ и моделей ОУ; выпол-

няется формальная и желательно изоморфная реализация «автоматной» спецификации и спецификации ФЭЗ с помощью управляющей программы; производится сертификация этой программы с использованием графа достижимых маркировок; строится комплексная программа — «управляющая программа—программа, моделирующая объекты управления»; выполняется сертификация комплексной программы с помощью графа достижимых маркировок.

Для моделирования и сертификации предложена программная «оболочка», позволяющая наблюдать при изменениях значений входных переменных не только значения выходных и временных переменных, но и значения каждой многозначной переменной, кодирующей все состояния каждого автомата. Это позволяет в любой момент времени определять его положение в пространстве состояний, что особенно важно при необходимости формирования одинаковых кортежей значений выходных переменных в различных точках пространства состояний.

При этом отметим, что если заданный словесный «алгоритм» реализуется одним графом переходов без флагов и умолчаний значений выходных переменных, то этот граф может использоваться в качестве теста для проверки программы, наблюдая не только за ее входными и выходными переменными, но и за ее внутренними состояниями, обеспечивая слежение только за одной внутренней многозначной переменной. При не только формальном, но и изоморфном переходе от графа переходов к тексту программы в тестировании нет необходимости, так как оно может быть заменено сверкой построенного графа переходов с текстом программы, а по графу переходов может быть проведена сертификация программы.

Если при программировании используется язык, отличный от выбранного на четвертой стадии, то эту стадию рассматривают как стадию моделирования алгоритма управления и дополняют ее собственно программированием в базисе языка «бортового» вычислителя. При этом программирование выполняется формально (автоматически или вручную) и желательно изоморфно по откорректированной системе взаимосвязанных графов переходов или по окончательному тексту программы на выбранном языке программирования, в базисе которого выполнено моделирование.

Методы программной реализации графов переходов в базисе выбранных представителей указанных в стандарте типов языков изложены в настоящей книге.

Сравнение языка SFC в форме диаграмм «Графсет» и графов переходов выполнено в гл. 12. При этом отметим, что графы переходов являются частным случаем SFC-диаграмм, в которых отсутствуют символы, отражающие параллелизм в одной компоненте, что позволяет в случае необходимости выполнять визуализацию ГП в среде программирования «Графсет». При этом отметим, что структура SFC-диаграмм и необходимость отражения параллелизма в одной компоненте ограничивают выбор структурных и алгоритмических моделей и методов их реализации, увеличивая число внутренних переменных и требуемый объем памяти программ.

Изображение параллельных процессов за счет использования системы взаимосвязанных графов переходов позволяет их реализовать существенно более компактно. Требования к построению графов переходов, пред-

ложенные в настоящей работе, позволяют их читать непосредственно, в то время как при чтении SFC-диаграмм требуется помнить, что обозначают те или иные символы действий и какие правила умолчаний применяются, тем более что даже для отображения одной диаграммы, включая описания условий и действий, приходится использовать более одного экрана. Для понимания спецификаций в обоих случаях целесообразно строить графы достижимых маркировок.

В качестве примера автоматизации четвертой стадии разработки программного обеспечения Б. П. Кузнецовым при участии автора (на базе материалов гл. 14) разработан транслятор, который по структурированному тексту программы на некотором ядре языка СИ строит программу на языке инструкций ALPro для ПЛК фирмы «FF-Automation» (Финляндия). При этом текст программы на языке СИ для алгоритмов логического управления в большинстве случаев строится в результате указанного выше перехода от графа переходов к конструкции `switch`.

Транслятор может работать в двух режимах, отличающихся типом используемых управляющих конструкций (IF или шаговый регистр). В первом случае имеется возможность обеспечить доступность каждого состояния автомата при любых значениях входных переменных для всех других автоматов системы, а во втором — это свойство не обеспечивается, но программы могут быть существенно короче, так как каждый шаговый регистр весьма эффективен для изоморфной реализации графа переходов.

Транслятор допускает использование макроопределений языка ALPro.

При реализации вычислительных алгоритмов логика их работы может описываться с помощью графов переходов и предварительно отлаживаться независимо от самих вычислений. Автоматическая трансляция арифметических выражений языка СИ резко снижает трудоемкость программной реализации на рассматриваемом языке инструкций алгоритмов непрерывного управления (включая регуляторы различных типов), которые предлагается предварительно отлаживать и моделировать на языке СИ.

Третья и четвертая стадии в рамках предлагаемой технологии, так же как и при применении SFC-диаграмм в ISaGRAF-продукте, могут быть скрыты от Пользователя, если разработанный программный продукт позволяет непосредственно транслировать систему взаимосвязанных графов переходов, отображаемых на дисплее, в программу на заданном языке программирования или непосредственно в объектный код.

Исходя из изложенного и работ [299, 305], по мнению автора, в стандарт должен быть введен еще один графический язык — СВГП, а также изложены методы программирования СВГП в базисе других языков, в том числе и описанных в стандарте, при использовании графов переходов в качестве языка спецификаций.

19.4. Технология автоматизации фирмы «Сименс» и SWITCH-технология

До последнего времени термин SIMATIC был синонимом ПЛК, выпускаемых фирмой «Сименс» [238].

В настоящее время этот термин должен пониматься значительно шире: SIMATIC это комплекс аппаратно-программных средств для построения широкого спектра систем автоматизации в различных отраслях промышленности.

Новое семейство средств автоматизации, дополняющее ПЛК семейства S5, является ответом на вопрос, возникший в последнее время как перед фирмой «Сименс», так и всеми другими фирмами, работающими в области автоматизации в промышленности: программируемые логические контроллеры ИЛИ промышленные компьютеры?

Ответ фирмы «Сименс» состоит в следующем: программируемые логические контроллеры И промышленные компьютеры [305].

В состав нового семейства средств автоматизации фирмы «Сименс» входят промышленные компьютеры SIMATIC M7-300 и SIMATIC M7-400 и программируемые логические контроллеры SIMATIC S7-200, SIMATIC S7-300, SIMATIC S7-400 и SIMATIC C7-620.

Появление новых аппаратных средств привело к совершенствованию имеющихся и разработке новых программных продуктов, предназначенных в том числе и для функционального программирования. Именно продукты этого назначения и рассматриваются в настоящем разделе.

Функциональное программное обеспечение для ПЛК фирмы «Сименс» построено на базе стандарта IEC 1131-3, рассмотренного в предыдущем разделе. Этот стандарт включен в Европейский стандарт EN6.1131-3. В Европе рассматриваемый стандарт был включен также и в национальные стандарты: DIN EN6.1131-3 (Германия) и BF EN6.1131-3 (Великобритания). При этом обратим внимание на тот факт, что в каталоге [305] в основном приводятся ссылки только на стандарт IEC 1131-3.

Язык функциональных схем («Function Block Diagram» — FBD), определенный стандартом, поддерживается библиотекой блоков STEP 7 следующих типов:

— организующие («Organization software blocks» — OB), которые осуществляют координацию работы блоков других типов в программе;

— функциональные («Function blocks» — FB) и системные функциональные блоки («System function blocks» — SFB), реализующие необходимые пользователю операции, включая типовые цифровые («Instance data blocks» — IDB) и системные цифровые блоки («System data blocks» — SDB);

— функции («Function» — FC), являющиеся подпрограммами наиболее часто используемыми пользователями;

— блоки данных («Data blocks» — DB), применяемые для запоминания данных Пользователя.

Языки «Операторный список» («Statement List» — STL) и «Лестничные схемы» («Ladder Diagram» — LAD) аналогичны языкам «Instruction List» (IL) и «Ladder Diagram» (LD), определенным стандартом.

Язык «S7-SCL» («Standart Control Language») является языком высокого уровня, базирующимся на языке «Паскаль». Язык «S7-SCL» аналогичен языку «Structured Text» (ST), определенному стандартом.

Язык «S7-Graph technology software» (аналог языка «Графсет») базируется на языке «Sequential Function Chart» (SFC), определенном стандартом.

В дополнение к языкам, описанным в стандарте, фирмой «Сименс» разработаны также и другие программные продукты.

Для программирования промышленных компьютеров SIMATIC M7 разработан программный продукт «M7-Pro C/C++».

Для программирования задач управления, описываемых с помощью нечеткой логики, разработан программный продукт «Fuzzy control software for special applications».

Разработан также программный продукт для построения управляющих систем из стандартных блоков для специальных применений («SIMATIC S7 — standard control system software for special applications»).

Язык непрерывных функциональных диаграмм («Continuous Functions Chart» — CFC) используется для объединения между собой блоков, написанных на языках STL, LAD или SCL для контроллеров SIMATIC S7, или стандартных библиотек (СИ программ) для компьютеров SIMATIC M7, специально предназначенных для описания непрерывных функций.

В ближайшее время должен появиться язык блок-схем для управляющих систем («Control System Flowchart» — CSF).

Наиболее интересным применительно к тематике настоящей книги является появление программного продукта «S7-HiGraph technology software», который базируется на диаграммах состояний (state diagram), что является другим названием графов переходов. При использовании диаграмм состояний каждая функциональная единица, соответствующая алгоритму управления одной из составляющих сложного объекта, описывается диаграммой состояний. Эти диаграммы могут «синхронизироваться» между собой посредством сообщений или головной диаграммы состояний. Это обеспечивает получение управляемых состояниями связанных автоматов, что повышает наглядность организации программ ПЛК. Этот подход во многом совпадает с подходом, предложенным и внедренным автором в 1991 году [167, 269] и описываемым в настоящей книге.

Интересным является то, что этот продукт имеет название с приставкой Hi («высокий»), что выделяет его среди имеющихся на мировом рынке других программных продуктов, предназначенных для управления. Настоящая книга также пытается убедить читателя в том, что графы переходов для Специалистов в области управления являются Hi-графами.

Удивительным является тот факт, что единственная диаграмма состояний, приведенная в качестве примера в [305], является незамкнутой, а вся диаграмма в целом, скорее, напоминает картинку, а не математическую модель, по которой возможно формальное программирование (как это предлагается в настоящей работе), так как формальное описание условий и действий в отличие от предлагаемого автором подхода выполняется отдельно на одном из языков, указанных выше (например, STL или LAD).

Из приведенного обзора следует, что в настоящее время фирма «Сименс» предоставляет широчайший спектр аппаратно-программных средств, определяющих ее технологию автоматизации промышленных объектов.

Недостатки предлагаемых программных продуктов применительно к задачам логического управления состоят в следующем:

— практически все программные продукты не связаны между собой и отсутствует единая методология их использования (например, программирование на языках СИ и СИ++ никак не связано с алгоритмизацией и программированием с помощью диаграмм состояний);

— отсутствуют формализованные методы построения корректных и «понятных» программ в базисе предложенных языков, т. е. предлагаемые средства позволяют записать на выбранном языке программу и транслировать ее, но не отвечают на вопрос: каким образом построить корректную и «понятную» программу?

— они могут применяться на ПЛК или ПК только фирмы «Сименс».

Предлагаемая автором SWITCH-технология позволяет для указанного класса задач устранить указанные недостатки, а настоящая книга в целом может явиться полезным пособием для пользователей ПЛК и ПК, выпускаемых различными фирмами мира, в том числе и фирмой «Сименс», так как дополняет стандарт IEC 1131 в части методов алгоритмизации и программирования.

19.5. Системы управления реального времени и SWITCH-технология

В настоящее время в промышленности используется большое число программных пакетов для создания интерфейса «человек-машина» («Man Machine Interfase» — MMI) и программного обеспечения операторских станций для распределенных систем управления («Destributed Control Systems» — DCS) технологическими процессами («Supervisor Control And Data Acquisition» — SCADA) [307].

Указанные программные продукты обеспечивают создание графического интерфейса Пользователя («Graphic User Interfase» — GUI) и являются основой для построения систем реального времени («Real Time System»).

Эти системы обычно строятся как локальные сети («Local Area Network» — LAN), в которых с помощью исполнительной системы («Runtime System») осуществляется динамический обмен данными («Dynamic Data Change» — DDE) между вычислительными устройствами, входящими в сеть.

При применении SCADA-систем типичным является следующая последовательность действий [307]:

— формирование статических изображений рабочего окна;

— формирование динамических объектов рабочего окна;

— описание алгоритмов отображения и управления;

— запуск программы монитора реального времени (программы «Runtime»).

В рамках настоящей работы наибольший интерес представляет вопрос: как в SCADA-системах выполняется алгоритмизация и программирование задач логического управления?

Общий ответ на этот вопрос состоит в следующем [307]. В простейшем случае при помощи текстового редактора на языке типа «BASIC» записываются логические и математические формулы с использованием логических имен переменных. В более сложных пакетах алгоритм может описываться при помощи языка функциональных блоков. При этом во многих системах предусматривается создание новых блоков, реализованных текстами программ или формулами, написанными на встроенном языке высокого уровня.

Рассмотрим более конкретно, в чем состоит ответ на сформулированный выше вопрос на примерах некоторых SCADA-систем.

Программное обеспечение «In Touch» фирмы «Wonderware» (США) [329] является объектно-ориентированным интерфейсом ММ1-систем. Эта фирма является пионером в использовании «Windows»-cHCTeM в промышленной автоматизации. Широкий набор драйверов позволяет применять этот пакет в промышленных компьютерах (Industrial Computer) и промышленных рабочих станциях (Industrial Workstation) [312], размещаемых на верхнем уровне систем управления, при использовании на нижнем уровне ПЛК.

Функции в системе поддерживают математические и логические формулы. Пользователь имеет возможность описывать и собственные функции, например на языке СИ.

Пакет «Genesis for Windows» (GFW) фирмы «Iconic» (США) создан на основе программы — ядра реального времени («Real Time Server» — RTS). В состав инструментальной части пакета GFW входит средство конфигурирования RTS при помощи графического языка функциональных блоков («Strategy Builder») [308, 309].

В пакете «Genie», предназначенном в основном для программной поддержки вычислительных устройств фирмы «Advantech» (США), используется аналогичный подход к реализации алгоритмов [307].

Пакет «TRASE MODE» фирмы «Ad Astra» (Россия) также предназначен для разработки, настройки и запуска в реальном времени систем управления [310, 311].

Логика организации обработки данных в этом пакете ориентирована на технологическое применение. При этом гибкость обработки обеспечивается сочетанием стандартных методов обработки, реализованных в системе, с нестандартными методами.

Если стандартных методов преобразования данных недостаточно, то можно воспользоваться интерпретатором языка пользовательских формул. В этих формулах наряду с арифметическими и логическими операциями могут применяться оператор присваивания и условный оператор.

В предлагаемом фирмой языке формул нет явных средств для организации циклов, которые необходимы для решения большого класса задач. Это объясняется цикличностью работы самого Монитора Реального Времени. Поэтому, если необходимо организовать циклы типа DO WHILE или FOR DO, требуется использовать условные операторы и формировать

по условиям значения «каналов», вызывающих соответствующие фрагменты файла формул.

Если в системе управления сложные расчеты невозможно выполнить ни с помощью стандартных методов преобразований, ни с помощью интерпретатора формул, имеется возможность применить математические модели, написанные на языке СИ. Они оформляются как резидентные средства и обмениваются с пакетом «TRASE MODE» через общую область памяти.

Изложенный подход может быть использован не только для вычислителей верхнего уровня, на котором из-за меньшей надежности по сравнению с вычислителями нижнего уровня управление в основном должно ограничиваться заданием параметров для устройств нижнего уровня, но и для IBM-совместимых контроллеров [312] нижнего уровня, на которых можно решать не только задачи управления, но и полноценные задачи визуализации процессов.

Поэтому в рамках пакета имеются Микромониторы Реального Времени для обеспечения управления на нижнем уровне при применении IBM-совместимых контроллеров.

Пакет реализует встроенные функции обмена данными с платами ввода-вывода контроллеров «Micro PC» фирмы «Octagon Systems» (США) [313]. Он позволяет организовать обмен данными с последовательными интерфейсами RS-232 и RS-485 по встроенным протоколам. Пакет обеспечивает обмен данными по сети на базе последовательного интерфейса с любыми ПЛК, поддерживающими обмен по протоколу MODBUS (например, для таких ПЛК, как «Modicon» и «Autolog»). Пакет позволяет осуществлять обмен данными с некоторыми другими типами ПЛК, а также с устройствами связи с объектом контроллеров ADAM серии 4000 фирмы «Advantech» [313]. Кроме того, возможен обмен данными с внешними устройствами с применением внешних программ связи — драйверов.

К программному обеспечению рассматриваемого типа относятся также пакеты «Paragon TNT» («Totally New Technology») фирмы «Intec Controls Corp.» (США) [314], «Lookout» фирмы «National Instruments» (США) [315] и «Mistic MMI» фирмы «Opto-22» (США) [234].

Пакет «Lookout» содержит встроенную библиотеку стандартных алгоритмов управления и обеспечивает возможность построения новых алгоритмов, а в пакете «Mistic MMI» программирование выполняется на основе блок-схем алгоритмов с помощью программы «Cyrano».

Кроме перечисленных разработаны и другие пакеты (описанные, например, в [308]), в том числе работающие под управлением операционной системы реального времени QNX фирмы «QNX Software Systems Ltd.» («QSSL») (Канада) [316]. Эта система является модульной и построена на базе микроядра, что позволяет приводить ее в соответствие с применяемыми вычислительными средствами.

К графическим пользовательским интерфейсам, работающим под управлением этой операционной системы, относится, например, такой пакет, как «RealFlex» фирмы «BJ Software Systems» («BJSS») (США) [233]. В состав этого пакета входит компилятор с языка управляющих последовательностей («Control Sequence Language» — CSL). Этот простой макроязык представляет возможность для математической, логической и

хронометрической обработки информации для использования в описаниях управляющих последовательностей.

Фирма «Cogent Real Time Systems Inc.» (США) обеспечила возможность совместного применения программного обеспечения «In Touch» и операционной системы QNX [317], а фирма «Corporate Headquaters» (Канада) разработала пакет «X Window System for QNX» [317].

Инструментальной средой визуального проектирования систем управления, разработанной на базе операционной системы QNX и графического «оконного» интерфейса «Photon» фирмы «QSSL» [318], спроектированного специально для применения во встроенных (embedded) системах управления, является пакет «RTWin», созданный фирмой «SWD Real Time Systems Ltd.» (Россия) [319].

Этот пакет позволяет строить пользовательские программы из блоков, часть из которых являются стандартными, а остальные могут быть программно реализованы на языке СИ. Трансляция и объединение в исполняемый код могут быть выполнены с помощью компилятора C/C++ фирмы «Watcom International Corp.» (США).

Другой инструментальной средой для разработки пользовательских программ, выполняемых под управлением операционной системы QNX, является пакет «Virtual Global Objects» («Virgo») фирмы «Alter Sys.» (Канада). Пакет обеспечивает «дружественность» процесса управления («Process Control Partner» — PCP) и обозначается как «PCP Virgo» [320].

Для построения пользовательских программ фирмой разработан язык «Fourth Generation Language (4GL)» [321], являющийся достаточно простым и мощным алгоритмическим языком высокого уровня, соответствующим требованиям, предъявляемым стандартом ШС 1131-3 к языкам типа «Structured Text». Этот язык наряду со стандартными функциональными возможностями языка высокого уровня, включающими в том числе и конструкцию switch, содержит большое число встроенных функций, упрощающих построение управляющих программ. При этом функции Пользователя могут писаться не только на языке 4GL, но и на языке ANSI C.

Из выполненного обзора следует, что ни в одном из перечисленных пакетов не поддерживается (программно или хотя бы методически) технология алгоритмизации и программирования, аналогичная предлагаемой автором. При этом Разработчик остается без методов наглядной алгоритмизации и формального программирования — «один на один» с функциональными блоками, блок-схемами алгоритмов, языками формул CSL, 4GL, СИ и т. д.

Более того, даже в тех случаях, когда в документации [322] приводится развернутый примере описанием пошагового использования SCADA-системы, почему-то подробно описываются все этапы работы с ней, за исключением реализации алгоритмов логического управления.

Отсутствие указанных методов не позволяет создавать понятное и надежное функциональное программное обеспечение для рассматриваемого в данной работе класса задач.

Из содержания настоящей книги следует, что изложенные в ней методы, объединенные понятием «SWITCH-технология», могут (по крайней мере методически) восполнить указанный пробел применительно практически ко всем языкам программирования, применяемым в SCADA-системах.

Общение автора с разработчиками одной из таких систем привело к тому, что SWITCH-технология начинает использоваться и в этой области (в SCADA-системах) — при разработке функциональных блоков, реализуемых программами на языке СИ, как это имеет место, например, для алгоритма «Metal Plant» в [319].

Предлагаемый подход может использоваться также в рамках интегрированных комплексов программ промышленной автоматизации, таких, например, как «Factory Suite» фирмы «Wonderware» [337]. Этот комплекс поддерживает и объединяет три уровня управления — Control, SCADA и MES («Manufacturing Execution System»).

Для первого уровня управления предназначен пакет «In Control», являющийся инструментальной системой программирования ПЛК с применением таких языков по стандарту IEC 1131-3, как лестничные схемы, последовательные функциональные диаграммы и структурный текст.

На втором уровне управления используются пакет «In Touch», рассмотренный выше, и пакет «In Support», обеспечивающий нахождение и устранение неисправностей, а также ведение технической документации.

На третьем уровне управления применяются пакет «In Track», предназначенный для построения систем управления дискретными производствами, и пакет «In Batch», обеспечивающий построение систем управления циклическими производствами, процедуры в котором описываются на языке последовательных функциональных диаграмм.

Рассмотренный комплекс тесно связан с пакетом «Back Office Suite» фирмы «Microsoft».

19.6. SWITCH-технология, программирование, автоматы и цепи Маркова

SWITCH-технология, излагаемая в настоящей работе, базируется на автоматном подходе, суть которого состоит во введении в явном виде в программирование основного понятия теории автоматов — «состояние» и в формальном и изоморфном построении программ по графам переходов, вершины которых предлагается кодировать многозначно.

При этом для обеспечения понятности программы переход в «новое» состояние должен однозначно зависеть только от «настоящего» состояния программы и входных воздействий, определяющих переход в новое состояние, и не должен зависеть от более «глубокой» предыстории.

Выходные воздействия, формируемые программой, также должны однозначно зависеть либо только от «настоящего» ее состояния, либо от этого состояния и входных воздействий, определяющих переход в «новое» состояние, и также не должны зависеть от более «глубокой» предыстории: при известном «настоящем» «будущее» не должно зависеть от «прошлого» [294].

Аналогичный подход применяется и в теории вероятностей. В [295] Б. В. Гнеденко пишет: «...математический анализ применим к исследованию процесса изменения поведения некоторой системы только в том случае, когда сделано предположение о том, что каждое возможное состояние этой системы вполне определено посредством некоторого

математического аппарата. С такой картиной мы встречаемся, например, в механике, когда предполагаем, что реальные движения систем материальных точек полностью могут быть описаны для любого момента времени указанием этого момента времени t и ее состоянием s в любой предыдущий момент времени t_0. При этом под состоянием системы в механике понимается задание положения точек материальной системы и их скоростей.

Вне классической механики, собственно во всей современной физике, приходится иметь дело с более сложным положением, когда знание состояния системы в какой-либо момент времени t_0 уже не определяет однозначно состояния системы в последующие моменты времени, а лишь определяет вероятность того, что система будет находиться в одном из состояний некоторого множества состояний системы.

Если дополнительное знание состояний системы в моменты $t < t_0$ не изменяет этой вероятности, то естественно назвать выделенный класс процессов процессами без последействия, или, за их аналогию с цепями Маркова, — процессами марковского типа».

Поэтому в качестве одной из основных моделей в теории вероятностей используются цепи Маркова, позволяющие при определении вероятностной меры рассматривать не любые последовательности случайных экспериментов, каждый из которых имеет конечное число возможных исходов, а только такие процессы, для которых исход данного опыта зависит от исхода предшествующего опыта и, более того, характер этой зависимости одинаков для всех этапов последовательности опытов [296].

При этом исходом каждого эксперимента служит один из конечного числа возможных исходов $s_1\ s_2,\ldots,\ s_n$, причем в каждом эксперименте вероятность исхода S_j либо вовсе не зависит от исходов предшествующих экспериментов, либо зависит только от исхода единственного эксперимента, непосредственно предшествующего данному. Эта зависимость задается числами P_{ij}, определяющими вероятность исхода s_j заданного эксперимента при условии, что предшествующий эксперимент имел исход S_i. Исходы $s_1,s_2,\ldots,s_n$ называются состояниями, а числа P_{ij} — вероятностью перехода из состояния с номером i в состояние с номером у.

Графическим способом представления вероятностей переходов является граф переходов.

Граф переходов цепи Маркова совпадает с графом переходов автономного вероятностного автомата без выходного преобразователя, и поэтому аппарат марковских цепей является основным при изучении этого класса автоматов [297].

Немарковские процессы [298] во многом напоминают автоматы с флагами, так как описывают зависимость следующего состояния не только от предыдущего состояния, но и от более «глубокой» предыстории. Сложность математического аппарата, возникающая по этой причине, ограничивает применение таких процессов.

Таким образом, обеспечение свойства независимости процессов от «глубокой» предыстории, резко упрощающего их понимание и описание и на котором базируется теория автоматов и теория марковских цепей, при использовании SWITCH-технологии возможно и в программировании. При этом отметим, что до настоящего времени устранению этой причины сложности понимания программ не уделялось должного внима-

ния, и можно надеяться, что настоящая работа приблизит к пониманию того, какими свойствами должны обладать легко «читаемые» программы, так как в настоящее время многие программы, применяемые на практике, могут быть условно (по аналогии) отнесены к классу немарковских.

Предлагаемая технология программирования может применяться не только при алгоритмизации и программировании задач «чисто» логического управления, но и при решении логико-вычислительных задач (Приложение 10), а также задач, связанных с представлением информации Оператору (Приложение 11). Программы, приведенные в этом Приложении, написанные С. Б. Терентьевым совместно с автором, могут быть разбиты на два класса: автоматно-событийные и событийно-автоматные. В этих названиях не совсем корректный термин «автоматный» заменяет термин «состояние» ввиду отсутствия в русском языке прилагательного от последнего термина.

В автоматах, реализуемых этими программами с помощью вложенных конструкций `switch`, используется многозначное кодирование состояний.

Первая программа первого класса реализует один автомат Мура, а вторая программа этого класса изоморфна автомату Мура, осуществляющему выбор формы представления информации, и комбинационному автомату исполнения.

Первая программа второго класса реализует декомпозированный (ввиду непротиворечивости входных переменных) единый автомат Мура в виде трех автоматов — двух последовательностных автоматов Мура и одного комбинационного автомата исполнения. Важнейшей особенностью этих автоматов Мура является то, что различные автоматы имеют общее пространство состояний: их состояния кодируются одной и той же переменной. В общем случае при наличии n непротиворечивых входных переменных автомат Мура всегда может быть декомпозирован на n автоматов с общим пространством состояний. В данном случае $n = 3$.

Вторая программа второго класса построена на базе предыдущей программы с учетом «чисто» последовательных переходов в каждом из автоматов Мура.

Из сопоставления программ первого и второго класса следует, что, несмотря на то что программы второго класса в данном случае более компактны, принцип их построения (первичность событий относительно состояний) не соответствует принципам работы Оператора при выборе формы представления информации, так как для Оператора первичным является текущее состояние автомата, описывающего функционирование меню, в зависимости от которого он нажимает соответствующую клавишу клавиатуры, что формирует входное событие. Несмотря на то что внешнее поведение программ первого и второго класса не отличается, применение событийно-автоматных программ обычно нецелесообразно, так как в этом случае нарушается изоморфизм между принципом их построения и принципом работы Оператора.

Указанный изоморфизм имеет место для автоматно-событийных программ, для которых состояния первичны относительно событий, что является естественным, так как состояния, на основе которых осуществляется управление, являются устойчивыми (статичными), а события по своей природе динамичны. Поэтому построение и использование таких

программ при отсутствии жестких ограничений на объем памяти и быстродействие, по мнению автора, более целесообразно. Аналогичный вывод можно сделать из рассмотрения примера, приведенного в разд. 13.4.2.

Таким образом, так как понятия «состояние» и «событие» входят в понятие «автомат», то на практике наряду или вместо событийно-управляемого программирования целесообразно применять автоматное программирование, которое вместе с другими подходами применяется в программируемых логических контроллерах [305], но не получило достаточного развития при программировании ПЭВМ, за исключением использования автоматов в [360—362].

При применении автоматного программирования построение алгоритмов и программ должно начинаться с формирования дешифратора состояний, а не событий.