УДК 681.3.06 . 62-507

А.А. ШАЛЫТО, д-р техн. наук, **Н.И. ТУККЕЛЬ**

АВТОМАТНЫЙ ПОДХОД К СОЗДАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СИСТЕМ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ И «РЕАКТИВНЫХ» СИСТЕМ

ВВЕДЕНИЕ

Современные системы логического управления состоят из двух составляющих: аппаратной и программной.

Особенность аппаратуры состоит в том, что она разрабатывается одними специалистами, а изготавливается (комплексируется) – другими. Это приводит к необходимости проводить разработку в форме проектирования, связанного с выпуском большого числа разнотипных схем и других конструкторских документов, соответствующих действующим стандартам и досконально отражающих все аспекты жизненного цикла аппаратуры. Поэтому даже через много лет эта часть систем при необходимости может быть сравнительно легко модифицирована, в том числе и другими специалистами.

Принципиально иная ситуация имеет место при создании программного обеспечения, так как оно разрабатывается и «изготавливается» одними и теми же специалистами. Поэтому обычно разработка программ не выполняется в форме проекта той же степени подробности, как это делается для аппаратуры, что часто приводит к значительным трудностям при необходимости их модификации.

Для систем логического управления при вводе входных воздействий по опросу, как это выполняется, например, в программируемых логических контроллерах, предложена SWITCH-технология, предназначенная для алгоритмизации и программирования задач логического управления [1, 2].

В этой технологии базовым является понятие «состояние». Добавляя к нему понятие «входное воздействие», естественным образом вводится понятие «автомат без выхода» (автомат без выхода = состояния + входные воздействия). После добавления понятия «выходное воздействие» эта формула приобретает вид: автомат = состояния + входные воздействия + выходные воздействия.

При этом соответствующий подход к программированию может быть назван «автоматным программированием», а процесс проектирования программ – «автоматным проектированием» [1].

Авторы применили SWITCH-технологию при разработке системы управления дизель-генератором, реализуемой на промышленном компьютере с операционной системой QNX, в которой управляющая программа выполняется как один процесс, а программа, моделирующая объект – как другой процесс.

При этом был создан **вариант** SWITCH-технологии для разработки программного обеспечения более широкого класса систем управления – «реак-

тивных» («reactive») систем, реагирующих на события [3–9]. Такие системы обычно реализуются на промышленных компьютерах, работающих под управлением операционных систем реального времени.

ОСОБЕННОСТИ ПРЕДЛАГАЕМОГО ВАРИАНТА ТЕХНОЛОГИИ

Предлагаемый 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;

– 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, содержащий только «простые» состояния (гиперсостояния [3, 4] не используются);

– расширена (по сравнению с [1]) 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 [9], 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 [10]);

– 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 автомата;

– 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 [9];

– 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, также могут 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;

– 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 [9];

– 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 [11];

– 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, не реентерабельны (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);

– 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;

– 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» [1];

– каждая входная переменная и каждое выходное воздействие также реализуются функцией, что 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 вызовов функций протоколирования в функции автоматов, входных и выходных воздействий 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 предлагается создавать и сдавать в архив документацию (по крайней мере, в электронном виде), имеющую 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; алгоритмы, например в виде графов переходов, и исходные тексты вспомогательных модулей и функций, реализующих входные переменные, обработчики событий и выходные воздействия; протоколы для сертификации программы, выполняющие роль контрольных примеров [12]; 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; для этого составляется перечень исполняемых модулей программы, в котором для каждого из них указывается значение циклической контрольной суммы, отражающей любое изменение в нем; 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.

ДОСТОИНСТВА ПРЕДЛАГАЕМОГО ВАРИАНТА ТЕХНОЛОГИИ

Предлагаемый вариант технологии обладает следующими **достоинствами**:

– в отличие от объектного моделирования [7, 9], 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, в том числе специалистами, не участвовавшими в проектировании;

– 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 [12] 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 в виде подпрограммы на выбранном языке программирования; указанный изоморфизм позволяет при необходимости решить обратную задачу [9] – 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 можно использовать 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 конструктивен, чем другие подходы, например изложенные в [13];

– порождаемый некоторыми событиями протокол или его часть является соответствующим сценарием; таким образом, сценарий строится автоматически при анализе программы, а не вручную при ее синтезе, как это предлагается делать при других подходах [7,9]; ручное построение всей совокупности сценариев и формальный синтез системонезависимой части программы по ним для задач со сложной логикой практически не осуществимы;

– предлагаемый подход не исключает интерактивной отладки и сертификации;

поведение системы взаимосвязанных автоматов является разновидностью коллективного поведения автоматов [14] и может использоваться при построении «многоагентных систем, состоящих из реактивных агентов» [15]

ЗАКЛЮЧЕНИЕ

Уверенность авторов в целесообразности применения предлагаемого подхода базируется, в частности, на том, что еще в 1966 году Э.Дейкстра предложил в [16] ввести так называемые переменные состояния, с помощью которых можно описывать состояния системы в любой момент времени, и использовал для этих целей целочисленные переменные. При этом им был поставлен вопрос 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 состояния и связывать процессы через эти переменные По мнению Э.Дейкстры, диаграммы состояний могут оказаться мощным 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.

Другой классик разработки программного обеспечения, Ф.Брукс, отметил, что «сложность служит причиной трудности перечисления, а тем более понимания всех возможных состояний программы, а отсюда возникает ее ненадежность. Сложность служит также источником невизуализуемых состояний, в которых нарушается система защит» [12].

Предлагаемая технология основана на априорном задании состояний и их визуализации, и поэтому авторы надеются, учитывая мнение о работе [1], высказанное в [17], что она, по крайней мере, для систем логического управления и «реактивных» систем является приближением к «**серебряной пуле**» [12] в части создания качественных программ, тем более что Ф.Брукс отозвался благосклонно только о подходе Д.Харела [3, 4], также основанном на применении автоматов, преимущество по сравнению с которым показано в [18].

Целесообразность применения SWITCH-технологии подтверждается также и тем, что 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 вдобавок с его помощью стали разрабатывать драйверы» [19].

Использование предлагаемой технологии подтверждает также следующее высказывание: «... то, что не специфицировано формально, не может быть проверено, а то, что не может быть проверено, не может быть безошибочным» [20].

Авторы надеются, что автоматный подход к проектированию программ в соответствии с принципом Оккама «не размножает сущности без необходимости» и обладает «минимализмом» [21], в отличие от подходов, излагаемых для «реактивных» систем в [4], а для общего случая – в [9].

Использование 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 изменений [22].

СПИСОК ЛИТЕРАТУРЫ

1. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления //Промышленные АСУ и контроллеры. 1999. №9. С.33-37.
3. *Harel D. et al.* STATEMATE: A working environment for the development of complex reactive systems //IEEE Trans. Eng. 1990. №4. P. 403–414.
4. *Harel D., Politi M.* Modeling reactive systems with statecharts. NY: McGraw-Hill, 1998.
5. *Карпов Ю.Г.* Теория алгоритмов и автоматов. Курс лекций. СПб.: СПбГТУ, Нестор, 1998.
6. *xjCharts.* Release 2.0. User's Manual. Experimental Object Technologies. 1999 95 p.
7. *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. С. 44–51.
8. *STATEFLOW* for use with Simmulink. User's guide. Version 1. MA: Math Works, Inc 1998.
9. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя. М.: ДМК, 2000.
10. *Затуливетер Ю.С. Халатян Т.Г.* Синтез общих алгоритмов по демонстрациям частных примеров (автоматная модель обобщения по примерам). М.: Ин-т проблем управления, 1997.
11. *Гудман С., Хидетниеми С.* Введение в разработку и анализ алгоритмов. М.: Мир, 1981.
12. *Брукс Ф.* Мифический человеко-месяц, или Как создаются программные системы. СПб.: Символ , 2000.
13. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Использование конечных автоматов для тестирования программ //Программирование. 2000. №2. С.12–28.
14. *Варшавский В.И.* Коллективное поведение автоматов. М.: Наука, 1973.
15. *Круглый* стол. «Парадигмы искусственного интеллекта» //Новости искусственного интеллекта. 1998. №3. С.140–161.
16. *Дейкстра Э.* Взаимодействие последовательных процессов /Языки программирования. М.: Мир, 1972. С.10–51.
17. *Герр Р.* Новый поворот // PC Magazine / Russian Edition. 1998. №10. С.88–90.
18. *Шалыто А.А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000.
19. *Кук Д., Урбан Д., Хамилтон С.* Unix и не только. Интервью с Кеном Томпсоном //Открытые системы. 1999. №4. С.35–47.
20. *Зайцев С.С.* Описание и реализация протоколов сетей ЭВМ. М.: Наука, 1989.
21. *Герр Р.* Отладка человечества // PC Magazine / Russian Edition. 2000. №5. С.90-91.
22. *Астров В.В., Василенко В.С., Тотьменинов Л.В.* Вопросы создания интегрированных систем управления ядерными энергетическими установками кораблей // Системы управления и обработки информации. СПб.: ФНПЦ «НПО «Аврора»». Науч.-техн. сб. 2000 Вып. 1. С.45–52.