УДК 681.3.06 : 62-507

SWITCH-ТЕХНОЛОГИЯ — АВТОМАТНЫЙ ПОДХОД К СОЗДАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ «РЕАКТИВНЫХ» СИСТЕМ

© 2000 г. А.А.Шалыто, Н.И.Туккель

Федеральный научно-производственный центр — ФГУП «НПО «Аврора»»,
Санкт-Петербургский государственный институт точной механики и оптики (технический университет)
Санкт-Петербург

На основе автоматного подхода предложена технология создания программного обеспечения «реактивных» систем, поддерживающая этапы изучения предметной области, анализа, проектирования, реализации, отладки, сертификации и документирования.

1. ВВЕДЕНИЕ

Современные системы 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].

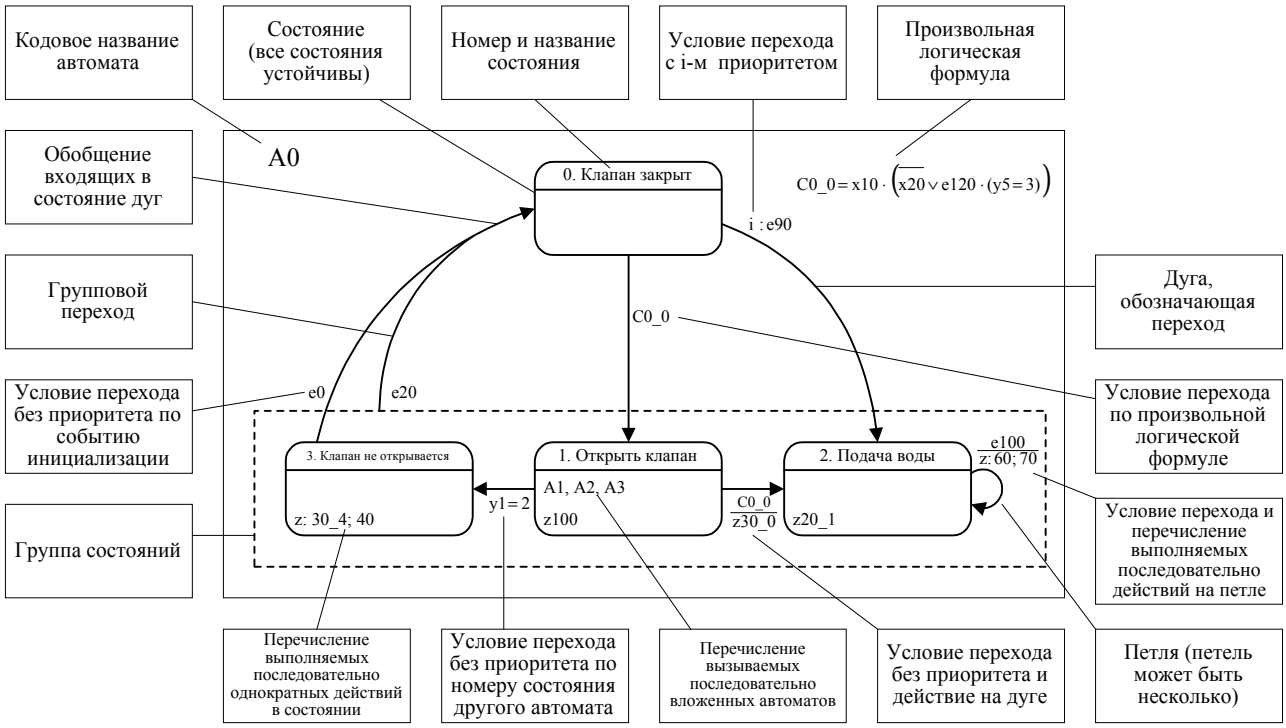
Авторы применили SWITCH-технологию при разработке системы управления дизель-генератором, реализуемой на промышленном компьютере с операционной системой QNX, в которой управляющая программа выполняется как один процесс, а программа, моделирующая объект — как другой процесс.

При этом был создан **вариант** SWITCH-технологии для разработки программного обеспечения более широкого класса систем управления — «реактивных» («reactive») систем, реагирующих на события [3-9]. Такие системы обычно реализуются на промышленных компьютерах, работающих под управлением операционных систем реального времени.

2. 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 (рис. 1);
- 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);



Используемые обозначения

An	автомат с номером n.
yn	переменная для хранения состояний автомата с номером n.
xj	входная переменная с номером j.
zk	выходное воздействие с номером k.
zd_m	m-ое значение выходного воздействия с номером d.
en	событие с номером n. В условиях переходов "en" является сокращенной записью предиката "e == n", а "$\overline{en}$" - сокращенной записью предиката "e != n".
yn = p	условие перехода по номеру состояния автомата с номером n (предикат "yn == p").
Cn_r	условие перехода с номером r для автомата с номером n.
i:	обозначение приоритета условия перехода (1 - наивысший приоритет).

Рис. 1. Нотация графов переходов

- **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]. Пример схемы взаимодействия автоматов приведен на рис. 2;
- 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;

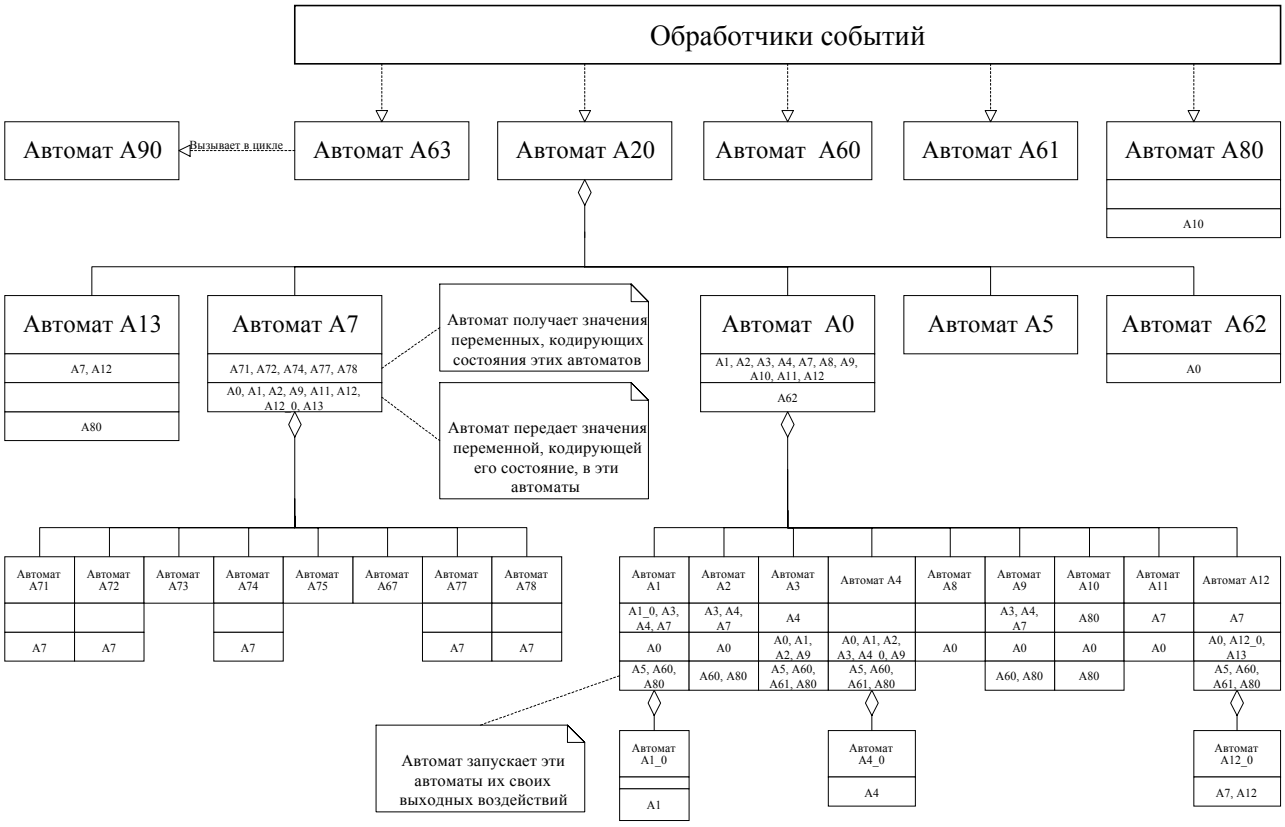


Рис. 2. Схема взаимодействия автоматов

- 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 [9];
- 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. Пример схемы связей автомата приведен на рис. 3;

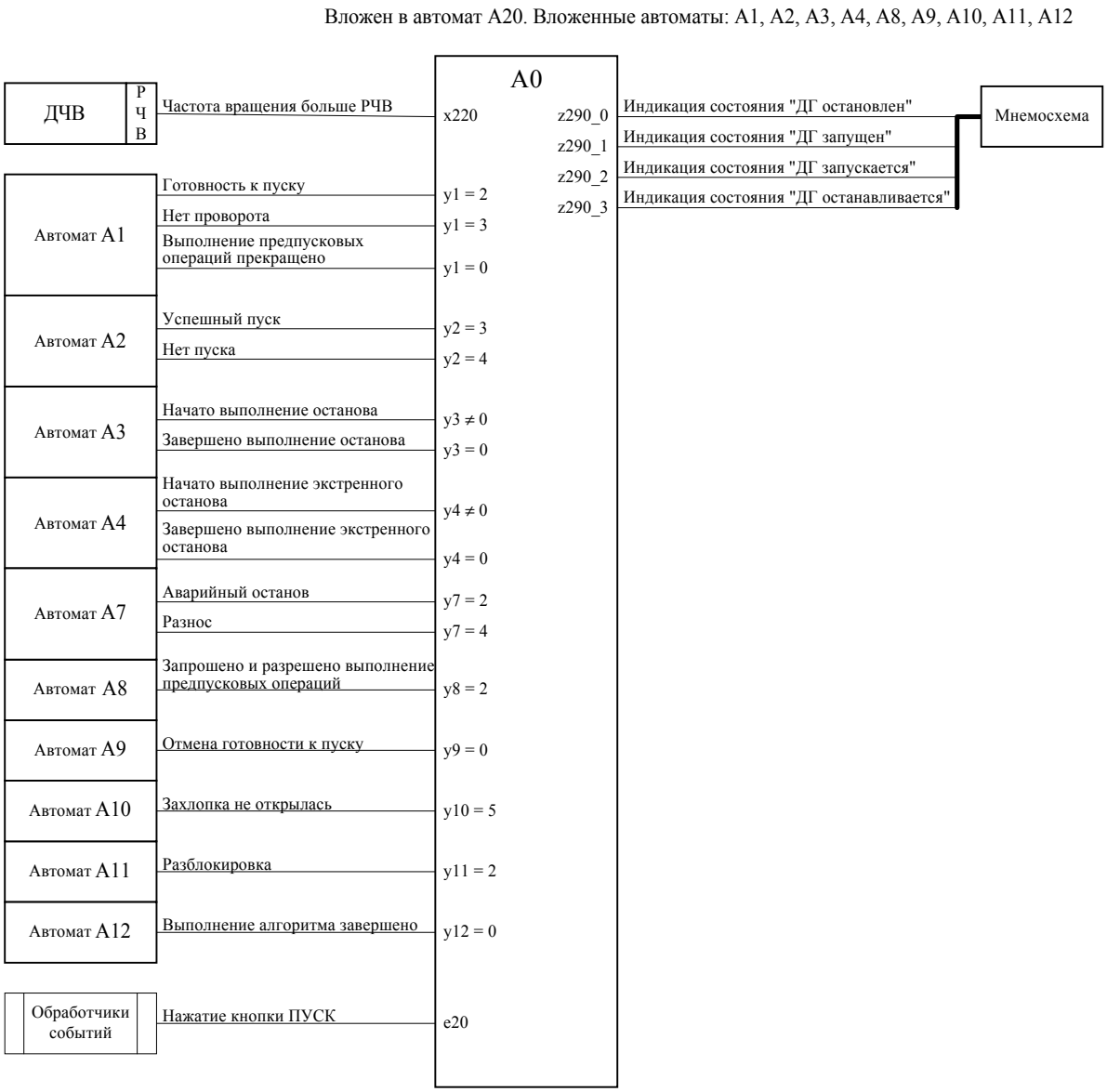


Рис. 3. Схема связей автомата

- 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. Пример графа переходов, соответствующий схеме связей, представленной на рис. 3, приведен на рис. 4. Для хранения состояний этого

автомата используется переменная **y0**, которая принимает девять значений (от 0 до 8);

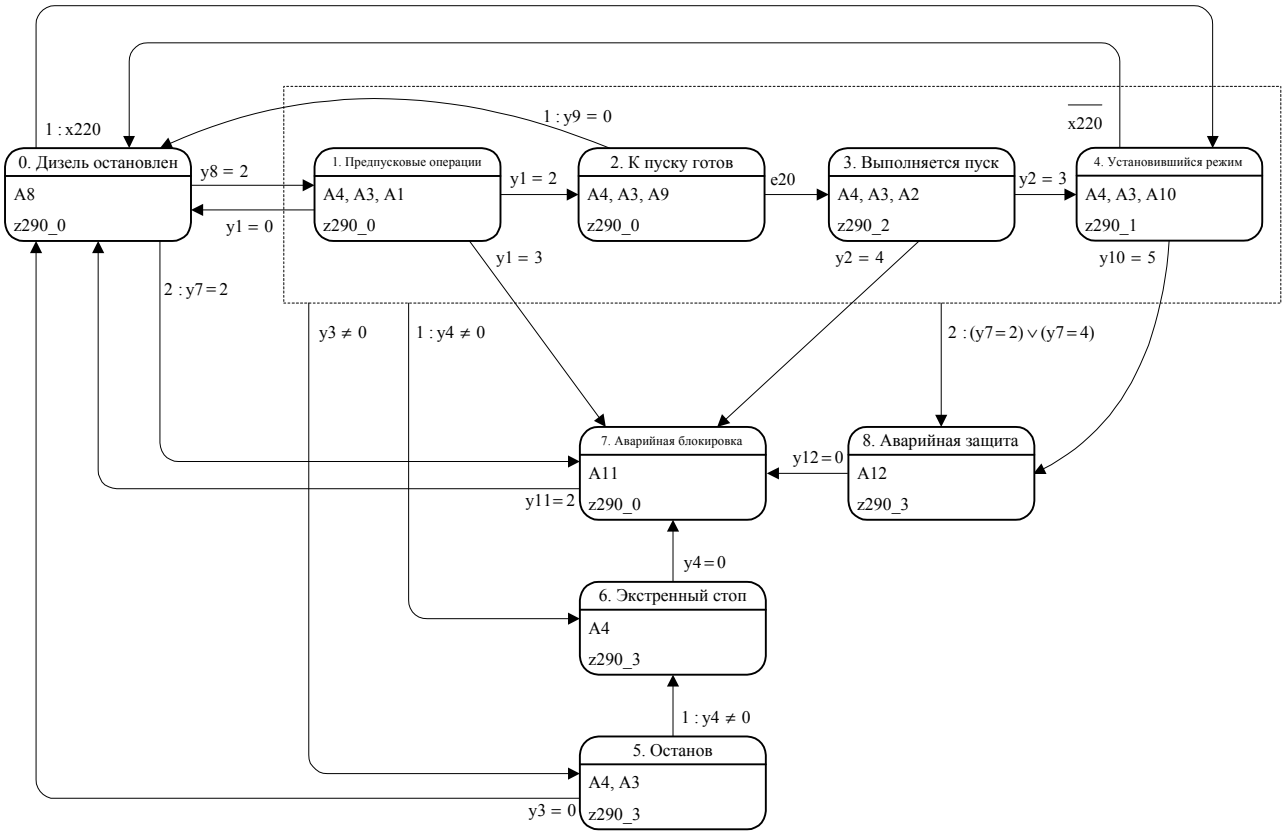


Рис. 4. Граф переходов

- 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, состоящие из наборов соответствующих функций (например, модуль управления таймерами);
- выделение функций декомпозирует задачу, а выделение состояний — ее последовательностную логику;
- обработчики событий содержат вызовы подпрограмм, реализующих графы переходов (автоматы), с передачей им соответствующих событий. Функции входных и выходных воздействий вызываются из подпрограмм, 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 (рис. 5);

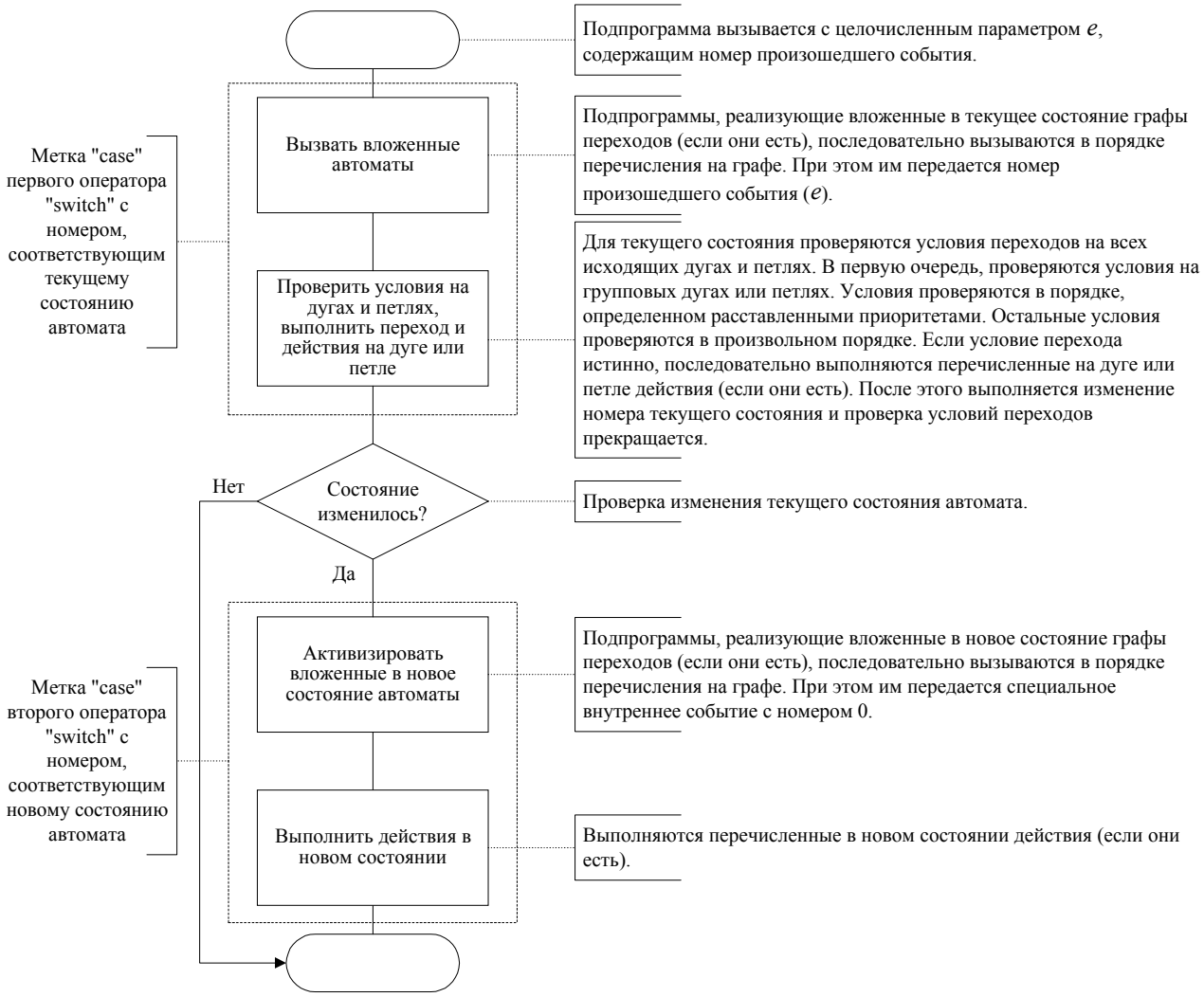


Рис. 5. Универсальный алгоритм реализации графов переходов

- каждый граф переходов **формально** и **изоморфно** реализуется отдельной функцией (подпрограммой), создаваемой по **шаблону** (Приложение 1), 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. Пример функции, реализующей автомат, представленный на рис. 4, приведен в Приложении 2;
- в общем случае для активизации автоматов, вложенных в новое состояние, они вызываются с событием e0;
- 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 реализуется 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 протоколирования;
- этап завершается построением **структурной схемы разработанного программного обеспечения**, отражающей взаимодействие его частей (рис. 6). Эта схема может включать схему взаимодействия автоматов, которая при этом отдельно не выпускается;

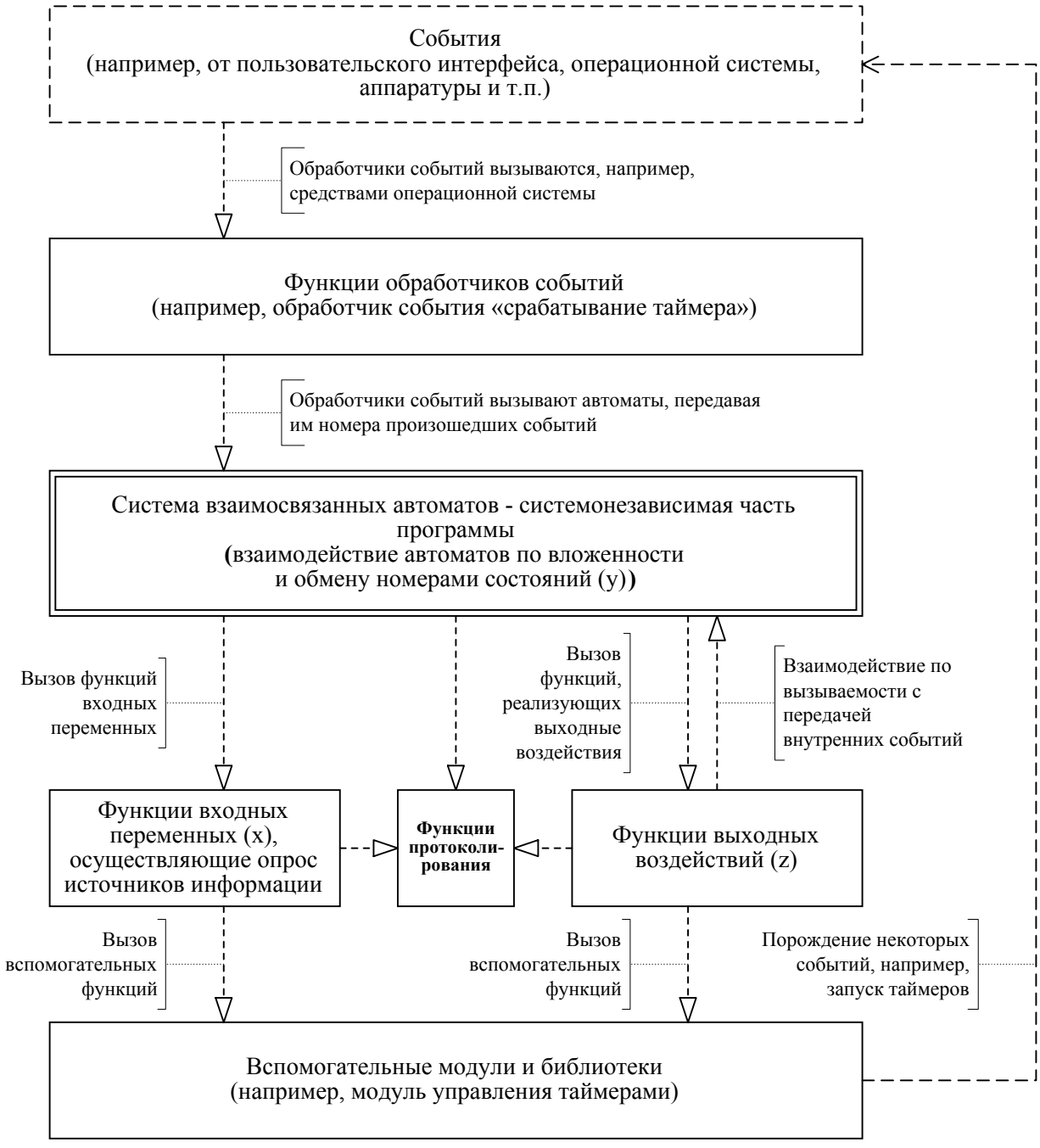


Рис. 6. Структурная схема событийных программ

- на **этапе отладки** обеспечена возможность одновременной индикации значений переменных состояний всех автоматов на одном экране;
- на **этапе сертификации** для каждой выбранной комбинации входных воздействий за счет введения вызовов функций протоколирования в функции автоматов, входных и выходных воздействий обеспечено **автоматическое ведение протокола** (история реакции на событие). 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. Примеры «полного» и «короткого» протоколов для одного из событий, обрабатываемого системой взаимодействующих автоматов (рис. 2), при выбранных значениях входных переменных, приведены в Приложении 3;
- 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.

3. 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 автоматов;
- 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], так как "постепенно среди теоретиков

программирования сложилось представление, что множество протоколов лучше характеризует программу, нежели сам исходный программный текст" [14];

- 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 [15] и может использоваться при построении «многоагентных систем, состоящих из реактивных агентов» [16].

4. ЗАКЛЮЧЕНИЕ

Уверенность авторов в целесообразности применения предлагаемого подхода базируется, в частности, на том, что еще в 1966 году Э. Дейкстра предложил в [17] 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 невизуализируемых 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 [18], 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 [19].

Целесообразность применения автоматного подхода 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**» [20].

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» [21].

Применение в предлагаемой парадигме понятия «автомат» в качестве центрального, соответствует его месту в теории управления, что принципиально отличает данный подход от других парадигм программирования.

СПИСОК ЛИТЕРАТУРЫ

1. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
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. 258 p.
5. *Карпов Ю.Г.* Теория алгоритмов и автоматов. Курс лекций. СПб.: СПбГТУ, Нестор, 1998. 129 с.
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. 477 p.
9. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя. М.: ДМК, 2000. 432 с.
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. 72 с.
11. *Гудман С., Хидетниеми С.* Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 366 с.
12. *Брукс Ф.* Мифический человеко-месяц или как создаются программные системы. СПб.: Символ, 2000. 304 с.
13. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Использование конечных автоматов для тестирования программ //Программирование. 2000. №2. С.12-28.
14. *Ершов А.П.* Смешанные вычисления //В мире науки. 1984. №6. С.28-42.
15. *Варшавский В.И.* Коллективное поведение автоматов. М.: Наука, 1973. 407 с.
16. *Круглый стол.* «Парадигмы искусственного интеллекта» //Новости искусственного интеллекта. 1998. №3. С.140-161.
17. *Дейкстра Э.* Взаимодействие последовательных процессов /Языки программирования. М.: Мир, 1972. С.9-86.
18. *Герр Р.* Новый поворот // PC Magazine / Russian Edition. 1998. №10. С.88-90.
19. *Шалыто А.А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.:Наука, 2000. 773 с.
20. *Кук Д., Урбан Д., Хамилтон С.* Unix и не только. Интервью с Кеном Томпсоном //Открытые системы. 1999. №4. С.35-47.
21. *Зайцев С.С.* Описание и реализация протоколов сетей ЭВМ. М.: Наука, 1989. 112 с.

ПРИЛОЖЕНИЕ 1

```
#include "photon_stuff.h"
#include "dg.h"
#include "log.h"
#include "defines.h"

void A_i_( int e, dg_t *dg )
{
  int y_old = dg->y_i_ ;

  // Протоколирование запуска автомата
  #ifdef A_i__BEGIN_LOGGING
    log_begin( dg, "A_i_", y_old, e ) ;
  #endif

  switch( dg->y_i_ )
  {
    case 0:
      // Вызвать вложенные автоматы
      // Проверить условия на дугах и петлях,
      // выполнить переход и действия на дуге или петле
    break ;

    ...

    case n:
      // Вызвать вложенные автоматы
      // Проверить условия на дугах и петлях,
      // выполнить переход и действия на дуге или петле
    break ;

    default:
      #ifdef A_i__ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "Ошибка в автомате A_i_: неизвестный номер состояния!", 0 ) ;
      #endif
  } ;

  // Проверка изменения текущего состояния автомата
  if( y_old == dg->y_i_ ) goto A_i__end ;

  {
    // Протоколирование перехода в автомате
    #ifdef A_i__TRANS_LOGGING
      log_trans( "A_i_", y_old, dg->y_i_ ) ;
    #endif
  } ;

  switch( dg->y_i_ )
  {
    case 0:
      // Произвести активизацию вложенных в новое состояние автоматов
      // Выполнить действия в новом состоянии
    break ;

    ...

    case n:
      // Произвести активизацию вложенных в новое состояние автоматов
      // Выполнить действия в новом состоянии
    break ;
  } ;

  // Протоколирование завершения работы автомата
  A_i__end: ;
  #ifdef A_i__END_LOGGING
    log_end( dg, "A_i_", dg->y_i_, e ) ;
  #endif
} ;
```

ПРИЛОЖЕНИЕ 2

```
#include "photon_stuff.h"
#include "dg.h"
#include "log.h"
#include "defines.h"

void A0( int e, dg_t *dg )
{
  int y_old = dg->y0 ;

  #ifdef A0_BEGIN_LOGGING
    log_begin( dg, "A0", y_old, e ) ;
  #endif

  switch( dg->y0 )
  {
    case 0:
      A8( e, dg ) ;
      if( x220(dg) )                    dg->y0 = 4 ;
      else
      if( dg->y7 == 2 )                 dg->y0 = 7 ;
      else
      if( dg->y8 == 2 )                 dg->y0 = 1 ;
    break ;

    case 1:
      A4( e, dg ) ; A3( e, dg ) ; A1( e, dg ) ;
      if( dg->y4 != 0 )                 dg->y0 = 6 ;
      else
      if( dg->y7 == 2 || dg->y7 == 4 )  dg->y0 = 8 ;
      else
      if( dg->y3 != 0 )                 dg->y0 = 5 ;
      else
      if( dg->y1 == 0 )                 dg->y0 = 0 ;
      else
      if( dg->y1 == 3 )                 dg->y0 = 7 ;
      else
      if( dg->y1 == 2 )                 dg->y0 = 2 ;
    break ;

    case 2:
      A4( e, dg ) ; A3( e, dg ) ; A9( e, dg ) ;
      if( dg->y4 != 0 )                 dg->y0 = 6 ;
      else
      if( dg->y7 == 2 || dg->y7 == 4 )  dg->y0 = 8 ;
      else
      if( dg->y3 != 0 )                 dg->y0 = 5 ;
      else
      if( dg->y9 == 0 )                 dg->y0 = 0 ;
      else
      if( e == 20 )                     dg->y0 = 3 ;
    break ;

    case 3:
      A4( e, dg ) ; A3( e, dg ) ; A2( e, dg ) ;
      if( dg->y4 != 0 )                 dg->y0 = 6 ;
      else
      if( dg->y7 == 2 || dg->y7 == 4 )  dg->y0 = 8 ;
      else
      if( dg->y3 != 0 )                 dg->y0 = 5 ;
      else
      if( dg->y2 == 4 )                 dg->y0 = 7 ;
      else
      if( dg->y2 == 3 )                 dg->y0 = 4 ;
    break ;

    case 4:
      A4( e, dg ) ; A3( e, dg ) ; A10( e, dg ) ;
      if( dg->y4 != 0 )                 dg->y0 = 6 ;
      else
      if( dg->y7 == 2 || dg->y7 == 4 )  dg->y0 = 8 ;
      else
      if( dg->y3 != 0 )                 dg->y0 = 5 ;
      else
      if( dg->y10 == 5 )                dg->y0 = 8 ;
```

```
    else
    if( !x220(dg) )                    dg->y0 = 0 ;
  break ;

  case 5:
    A4( e, dg ) ; A3( e, dg ) ;
    if( dg->y4 != 0 )                  dg->y0 = 6 ;
    else
    if( dg->y3 == 0 )                  dg->y0 = 0 ;
  break ;

  case 6:
    A4( e, dg ) ;
    if( dg->y4 == 0 )                  dg->y0 = 7 ;
  break ;

  case 7:
    A11( e, dg ) ;
    if( dg->y11 == 2 )                 dg->y0 = 0 ;
  break ;

  case 8:
    A12( e, dg ) ;
    if( dg->y12 == 0 )                 dg->y0 = 7 ;
  break ;

  default:
    #ifdef A0_ERRORS_LOGGING
      log_write( LOG_GRAPH_ERROR, dg->number,
                 "Ошибка в автомате A0: неизвестный номер состояния!", 0 ) ;
    #endif
} ;

if( y_old == dg->y0 ) goto A0_end ;

{
  #ifdef A0_TRANS_LOGGING
    log_trans( "A0", y_old, dg->y0 ) ;
  #endif

  #ifdef DEBUG_FRAME
    update_debug() ;
  #endif
} ;

switch( dg->y0 )
{
  case 0:
    A8( 0, dg ) ;
    z290_0(dg) ;
  break ;

  case 1:
    A4( 0, dg ) ; A3( 0, dg ) ; A1( 0, dg ) ;
    z290_0(dg) ;
  break ;

  case 2:
    A4( 0, dg ) ; A3( 0, dg ) ; A9( 0, dg ) ;
    z290_0(dg) ;
  break ;

  case 3:
    A4( 0, dg ) ; A3( 0, dg ) ; A2( 0, dg ) ;
    z290_2(dg) ;
  break ;

  case 4:
    A4( 0, dg ) ; A3( 0, dg ) ; A10( 0, dg ) ;
    z290_1(dg) ;
  break ;

  case 5:
    A4( 0, dg ) ; A3( 0, dg ) ;
    z290_3(dg) ;
  break ;
```

```
    case 6:
      A4( 0, dg ) ;
      z290_3(dg) ;
    break ;

    case 7:
      A11( 0, dg ) ;
      z290_0(dg) ;
    break ;

    case 8:
      A12( 0, dg ) ;
      z290_3(dg) ;
    break ;
  } ;

  A0_end: ;
  #ifdef A0_END_LOGGING
    log_end( dg, "A0", dg->y0, e ) ;
  #endif
} ;
```

ПРИЛОЖЕНИЕ 3

Полный протокол обработки нажатия кнопки ПОДГОТОВКА К ПУСКУ (событие **e10**) по основному сценарию, полученный автоматически с выделенными вручную этапами.

Нажатие кнопки ПОДГОТОВКА К ПУСКУ:

```
11:34:02.507{ DG1: A20: в состоянии 2 запущен с событием e10
11:34:02.507{ DG1: A7: в состоянии 0 запущен с событием e10
11:34:02.507{ DG1: A71: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x320 - температура масла меньше Тмм - вернул 0
11:34:02.507> DG1: x330 - температура масла больше Тмпр - вернул 0
11:34:02.507} DG1: A71: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A72: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x220 - частота вращения больше РЧВ - вернул 0
11:34:02.507} DG1: A72: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A73: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x220 - частота вращения больше РЧВ - вернул 0
11:34:02.507} DG1: A73: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A74: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x430 - температура воды меньше Твм - вернул 0
11:34:02.507> DG1: x440 - температура воды больше Твпр - вернул 0
11:34:02.507} DG1: A74: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A75: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x460 - давление наддува больше Рнпр - вернул 0
11:34:02.507} DG1: A75: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A76: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x470 - температура газов больше Твг - вернул 0
11:34:02.507} DG1: A76: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A77: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x480 - давление в отсеке ниже Ротс - вернул 0
11:34:02.507} DG1: A77: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A78: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x500 - сигнал ВОДА В ОХЛАДИТЕЛЕ - вернул 0
11:34:02.507} DG1: A78: завершил обработку события e10 в состоянии 0
11:34:02.507> DG1: x20 - защита включена - вернул 1
11:34:02.507> DG1: x20 - защита включена - вернул 1
11:34:02.507> DG1: x260 - частота вращения больше ППРЕД - вернул 0
11:34:02.507} DG1: A7: завершил обработку события e10 в состоянии 0
11:34:02.507{ DG1: A0: в состоянии 0 запущен с событием e10
11:34:02.507{ DG1: A8: в состоянии 0 запущен с событием e10
11:34:02.507> DG1: x220 - частота вращения больше РЧВ - вернул 0
11:34:02.507> DG1: x500 - сигнал ВОДА В ОХЛАДИТЕЛЕ - вернул 0
11:34:02.507> DG1: x510 - сигнал МВП НЕ ОТКЛЮЧЕН - вернул 0
11:34:02.507> DG1: x520 - сигнал ПРЕДЕЛЬНЫЙ ВЫКЛЮЧАТЕЛЬ - вернул 0
11:34:02.507> DG1: x700 - сигнал БЛОКИРОВКА ПУСКА из СУ ОКС - вернул 0
11:34:02.507T DG1: A8: перешел из состояния 0 в состояние 2
11:34:02.507} DG1: A8: завершил обработку события e10 в состоянии 2
11:34:02.507T DG1: A0: перешел из состояния 0 в состояние 1
11:34:02.507{ DG1: A4: в состоянии 0 запущен с событием e0
11:34:02.507{ DG1: A4_0: в состоянии 0 запущен с событием e0
11:34:02.507} DG1: A4_0: завершил обработку события e0 в состоянии 0
```

```
11:34:02.507} DG1: A4: завершил обработку события e0 в состоянии 0
11:34:02.507{ DG1: A3: в состоянии 0 запущен с событием e0
11:34:02.507} DG1: A3: завершил обработку события e0 в состоянии 0
11:34:02.507{ DG1: A1: в состоянии 0 запущен с событием e0
11:34:02.507T DG1: A1: перешел из состояния 0 в состояние 1
11:34:02.507{ DG1: A1_0: в состоянии 0 запущен с событием e0
11:34:02.507* DG1: z800_0. Сбросить счетчик проворотов.
11:34:02.507} DG1: A1_0: завершил обработку события e0 в состоянии 0
11:34:02.507* DG1: z10_1. Включить табло ПОДГОТОВКА К ПУСКУ.
11:34:02.507* DG1: z20_1. Включить табло ПРОКАЧКА МАСЛОМ.
11:34:02.507* DG1: z30_1. Включить табло ПРОВОРОТ.
```

Запустить МПН:

```
11:34:02.507* DG1: z400_1. Включить магнитный пускатель МПН.
11:34:02.507{ DG1: A80( z400 ): в состоянии 0 запущен с событием e410
11:34:02.507> DG1: x900( z400 ) - клапан открыт - вернул 0
11:34:02.507T DG1: A80( z400 ): перешел из состояния 0 в состояние 3
11:34:02.507* DG1: z270_2( z400 ). Индикация состояния клапана - закрывается/открывается.
11:34:02.507* DG1: z950_1( z400 ). Запустить таймер контроля срабатывания.
11:34:02.507* DG1: z1000_1( z400 ). Открыть/запустить.
11:34:02.507} DG1: A80( z400 ): завершил обработку события e410 в состоянии 3

11:34:02.507* DG1: z420_1. Подать питание на электромагнит стопа РЧВ.
```

Открыть клапан подачи воздуха к ДГ:

```
11:34:02.507* DG1: z430_1. Открыть клапан подачи воздуха к ДГ.
11:34:02.507{ DG1: A80( z430 ): в состоянии 0 запущен с событием e410
11:34:02.507> DG1: x900( z430 ) - клапан открыт - вернул 0
11:34:02.507T DG1: A80( z430 ): перешел из состояния 0 в состояние 3
11:34:02.507* DG1: z270_2( z430 ). Индикация состояния клапана - закрывается/открывается.
11:34:02.507* DG1: z950_1( z430 ). Запустить таймер контроля срабатывания.
11:34:02.507* DG1: z1000_1( z430 ). Открыть/запустить.
11:34:02.507} DG1: A80( z430 ): завершил обработку события e410 в состоянии 3
```

Открыть клапан подачи воздуха низкого давления:

```
11:34:02.507* DG1: z440_1. Открыть клапан подачи воздуха низкого давления.
11:34:02.507{ DG1: A80( z440 ): в состоянии 0 запущен с событием e410
11:34:02.507> DG1: x900( z440 ) - клапан открыт - вернул 0
11:34:02.507T DG1: A80( z440 ): перешел из состояния 0 в состояние 3
11:34:02.507* DG1: z270_2( z440 ). Индикация состояния клапана - закрывается/открывается.
11:34:02.507* DG1: z950_1( z440 ). Запустить таймер контроля срабатывания.
11:34:02.507* DG1: z1000_1( z440 ). Открыть/запустить.
11:34:02.507} DG1: A80( z440 ): завершил обработку события e410 в состоянии 3
```

Выдать команду на установку частоты вращения первой ступени:

```
11:34:02.507* DG1: z460. Выдать команду на установку частоты вращения первой ступени.
11:34:02.507{ DG1: A5: в состоянии 0 запущен с событием e380
11:34:02.507> DG1: x810 - заданная частота вращения выше установленной - вернул 1
11:34:02.507T DG1: A5: перешел из состояния 0 в состояние 2
11:34:02.507* DG1: z610_1. Увеличить установленную частоту РЧВ.
11:34:02.507} DG1: A5: завершил обработку события e380 в состоянии 2

11:34:02.507* DG1: z810_1. Запустить таймер контроля проворота.
11:34:02.507} DG1: A1: завершил обработку события e0 в состоянии 1
11:34:02.507* DG1: z290_0. Индикация состояния ДГ - остановлен.
11:34:02.507} DG1: A0: завершил обработку события e10 в состоянии 1
11:34:02.507{ DG1: A5: в состоянии 2 запущен с событием e10
11:34:02.507> DG1: x810 - заданная частота вращения выше установленной - вернул 1
11:34:02.517> DG1: x820 - заданная частота вращения ниже установленной - вернул 0
11:34:02.517} DG1: A5: завершил обработку события e10 в состоянии 2
11:34:02.517{ DG1: A13: в состоянии 0 запущен с событием e10
11:34:02.517} DG1: A13: завершил обработку события e10 в состоянии 0
11:34:02.517{ DG1: A62: в состоянии 0 запущен с событием e10
11:34:02.517T DG1: A62: перешел из состояния 0 в состояние 1
11:34:02.517* DG1: z1110_1. Запустить таймер регистрации параметров.
11:34:02.517* DG1: z1120_1. Запустить таймер индикации параметров.
11:34:02.517} DG1: A62: завершил обработку события e10 в состоянии 1
11:34:02.517} DG1: A20: завершил обработку события e10 в состоянии 2
```

Короткий протокол обработки нажатия кнопки ПОДГОТОВКА К ПУСКУ (событие **e10**) по основному сценарию.

```
11:41:06.188{ DG1: A20: в состоянии 2 запущен с событием e10
11:41:06.188* DG1: z800_0. Сбросить счетчик проворотов.
11:41:06.188* DG1: z10_1. Включить табло ПОДГОТОВКА К ПУСКУ.
11:41:06.188* DG1: z20_1. Включить табло ПРОКАЧКА МАСЛОМ.
11:41:06.188* DG1: z30_1. Включить табло ПРОВОРОТ.
11:41:06.188* DG1: z400_1. Включить магнитный пускатель МПН.
11:41:06.188* DG1: z270_2( z400 ). Индикация состояния клапана - закрывается/открывается.
11:41:06.188* DG1: z950_1( z400 ). Запустить таймер контроля срабатывания.
11:41:06.188* DG1: z1000_1( z400 ). Открыть/запустить.
11:41:06.188* DG1: z420_1. Подать питание на электромагнит стопа РЧВ.
11:41:06.188* DG1: z430_1. Открыть клапан подачи воздуха к ДГ.
11:41:06.188* DG1: z270_2( z430 ). Индикация состояния клапана - закрывается/открывается.
11:41:06.188* DG1: z950_1( z430 ). Запустить таймер контроля срабатывания.
11:41:06.188* DG1: z1000_1( z430 ). Открыть/запустить.
11:41:06.188* DG1: z440_1. Открыть клапан подачи воздуха низкого давления.
11:41:06.188* DG1: z270_2( z440 ). Индикация состояния клапана - закрывается/открывается.
11:41:06.188* DG1: z950_1( z440 ). Запустить таймер контроля срабатывания.
11:41:06.188* DG1: z1000_1( z440 ). Открыть/запустить.
11:41:06.188* DG1: z460. Выдать команду на установку частоты вращения первой ступени.
11:41:06.188* DG1: z610_1. Увеличить установленную частоту РЧВ.
11:41:06.188* DG1: z810_1. Запустить таймер контроля проворота.
11:41:06.188* DG1: z290_0. Индикация состояния ДГ - остановлен.
11:41:06.188* DG1: z1110_1. Запустить таймер регистрации параметров.
11:41:06.188* DG1: z1120_1. Запустить таймер индикации параметров.
11:41:06.188} DG1: A20: завершил обработку события e10 в состоянии 2
```