МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
ТОЧНОЙ МЕХАНИКИ И ОПТИКИ (ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

УДК 681.3.06: 62—507
ВГК ОКП
№ регистрационный 01.20.00 13546
Инв. №

"УТВЕРЖДАЮ"
Ректор СПбГИТМО (ТУ)
докт. техн. наук, профессор

_____ Васильев В.Н.

ОТЧЕТ

по научно-исследовательской работе № 10038
"Разработка технологии создания программного обеспечения систем управления на основе автоматного подхода"

Этап 2

"Разработка основных положений создания программного обеспечения "реактивных" систем"

Руководитель НИР
докт. техн. наук,
профессор,
заведующий кафедры
"Информационные системы"
Шалыто А.А.

Санкт-Петербург
2001

Список основных исполнителей

Руководитель НИР

Доктор технических наук, профессор, заведующий кафедры "Информационные системы"	Шалыто А.А.

Исполнители

Инженер-программист	Туккель Н.И.
Аспирант	Зинчик А.А.
Аспирант	Штенников Д.Г.
Аспирант	Тибет Э.М.
Аспирант	Шамгунов Н.Н.
Студент группы 639	Казаков М.А.
Студент группы 538	Макаров Н.В.
Студент группы 439	Корнеев Г.А.
Студент группы 438	Станкевич А.С.
Студент группы 338	Наумов Л.А.
Студент группы 338	Прокушкин И.А.

Реферат

Отчет содержит 144 страниц, 50 рисунков, 20 источников литературы.

Система управления, "реактивная" система, событийная система, состояние, событие, автомат, граф переходов, протокол, алгоритмизация, автоматное программирование, проектирование программ

Целью настоящей работы является разработка основных положений технологии создания программного обеспечения "реактивных" (событийных) систем. Эта технология поддерживает для рассматриваемого класса систем все этапы создания программного обеспечения, к которым относятся: изучение предметной области, анализ, проектирование, реализация, отладка, сертификация и документирование.

Расширено по отношению к системам логического управления понятие "входное воздействие" за счет использования "событий", которые в отличие от "входных переменных", не опрашиваются программой, а вызывают соответствующие им обработчики. Расширено также понятие "выходное воздействие" за счет перехода от двоичных переменных к произвольным подпрограммам (функциям). Дополнена нотация, применяемая при построении графов переходов, например, за счет перечисления вложенных автоматов.

При использовании предлагаемой технологии в отличие от объектно-ориентированного проектирования программ построение всех основных моделей основано на применении только автоматной терминологии, а для описания динамики используется модель только одного типа — система взаимосвязанных графов переходов. Использование графов переходов в качестве языка спецификации делает обозримым

даже весьма сложное поведение программы и позволяет легко вносить изменения как в спецификацию, так и в ее реализацию.

Разработан универсальный алгоритм программной реализации иерархии графов переходов для произвольного их количества и произвольного уровня вложенности. Каждый граф переходов формально и изоморфно реализуется по шаблону в виде подпрограммы на выбранном языке программирования.

Автоматическое ведение протокола в терминах спецификации (автоматов) обеспечивает возможность сертификации программы. Для сертификации в терминах, понятных Заказчику, могут применяться "короткие" протоколы, в которых указываются только формируемые выходные воздействия.

Подробное документирование проекта создания программного обеспечения упрощает внесение изменений в него даже через длительный срок после выпуска, в том числе и специалистами, не участвовавшими в проектировании.

Содержание

Введение 7
1. Классификация задач, решаемых с использованием автоматного подхода 14
1.1. Системы логического управления.....16
1.1.1. Реализация на программируемых логических контроллерах...16
1.1.2. Моделирование контроллера в однозадачных операционных системах 18
1.1.3. Моделирование контроллера в многозадачных операционных системах 19
1.2. "Реактивные" (событийные) системы.....22
2. Автоматы 26
2.1. Нотация схемы связей.....27
2.2. Нотация графов переходов.....30
2.3. Нотация схемы взаимодействия автоматов.....36
2.4. Модель автомата.....38
2.4.1. Запуск автомата.....38
2.4.2. Входные воздействия и переходы.....39
2.4.3. Выходные воздействия.....40
2.4.4. Вложенные автоматы.....40
3. Создание программного обеспечения событийных систем 43
3.1. Предлагаемая технология.....43
3.1.1. Изучение предметной области.....44
3.1.2. Проектирование.....44
3.1.3. Реализация.....45
3.1.4. Отладка и сертификация.....50
3.1.5. Документирование.....52
4. Практическое применение предлагаемой технологии 53
4.1. Сравнение традиционного подхода с предлагаемым.....53
4.2. Подсистема управления печатью.....64
4.2.1. Автомат контроля режима печати.....66
4.2.2. Автомат буферизации предварительного просмотра.....69
4.2.3. Автомат вывода на печать.....73
4.2.4. Системозависимая часть модуля печати.....76
4.2.5. Системозависимая часть менеджера печати.....81
4.2.6. Протоколы функционирования модуля печати.....85
4.3. Система управления судовым дизель-генератором.....87

4.3.1. Документирование процесса проектирования системы управления 88
4.3.2. Автомат включения-отключения системы управления 90
4.3.3. Автомат переключения режимов работы системы управления .. 93
4.3.4. Автомат предпусковых операций 98
4.3.5. Автомат контроля проворота 104
4.3.6. Автомат останова 106
4.3.7. Автомат ожидания разрешения выполнения предпусковых операций 110
4.3.8. Автомат аварийной и предупредительной сигнализации 112
4.3.9. Автомат контроля температуры масла 115
4.3.10. Автомат контроля давления масла 118
4.3.11. Протоколы функционирования системы управления 120
4.4. Программный имитатор дизель-генератора 125
4.4.1. Автомат имитации исполнительного устройства 126
4.4.2. Автомат имитации регулятора частоты вращения 128
4.4.3. Автомат имитации дизель-генератора 129
4.4.4. Автомат имитации изменения частоты вращения 132
5. Заключение 134
6. Публикации по результатам этапа 139
Список литературы 142
Глоссарий 143

Введение

В отчете излагаются основные положения технологии создания программного обеспечения "реактивных" ("reactive") систем, реагирующих на события. Системы этого класса могут быть также названы событийными. Эта технология базируется на основе теории автоматов.

Отчет создан в ходе выполнения работ по теме "Разработка технологии создания программного обеспечения систем управления на основе автоматного подхода" (регистрационный номер темы во Всероссийском научно-техническом центре (ВНТИЦ) - 01.20.00 13546), второй этап которой, выполненный в 2001 г., имеет наименование "Разработка основных положений технологии создания программного обеспечения реактивных систем".

Выполненная работа является продолжением исследований, проведенных в 2000 г. по первому этапу темы: "Разработка основных положений технологии создания программного обеспечения систем логического управления" (инвентарный номер отчета во ВНТИЦ - 02.2.00 102948), в рамках которого была разработана SWITCH-технология, предназначенная для алгоритмизации и программирования задач логического управления.

Перечислим основные особенности этой технологии:

– четко выделен этап проектирования, предшествующий этапу реализации. При этом строятся схемы и диаграммы, которые в дальнейшем формально и изоморфно преобразуются в текст программы. Такая организация процесса разработки аналогична принятой при создании аппаратуры, изготовление которой выполняется только по проектной документации;

– опыт использования предлагаемой технологии показывает, что при тщательном проектировании построенная на этапе реализации программа либо сразу работает

правильно, либо требует существенно меньшего времени на отладку, чем при традиционном подходе;

– использование понятия "внутреннее состояние" (в дальнейшем "состояние") в качестве центрального;

– состояния рассматриваются в качестве абстракций, которые применяются при алгоритмизации в явном виде, что позволяет дать следующее название предлагаемому подходу: "программирование с явным выделением состояний";

– понятие "состояние" совместно с понятием "входное воздействие" порождает понятие "автомат без выхода", а после введения понятия "выходное воздействие" - понятие "автомат" (конечный, детерминированный);

– использование понятия "автомат" в качестве базового в таких понятиях как "автоматное программирование" и "автоматное проектирование программ";

– применение автоматов в качестве основной модели при алгоритмизации и программировании задач логического управления;

– введено понятие "управляющий автомат", под которым понимается совокупность автомата и функциональных элементов задержки, с которыми автомат взаимодействует также как с объектами управления;

– для описания интерфейса каждого автомата разрабатывается схема его связей, содержащая в общем случае источники информации, автомат, функциональные элементы задержки, исполнительные механизмы объектов управления и средства представления информации, что позволяет отказаться от словесных обозначений на графе переходов автомата и перейти к символам соответствующих переменных;

– в качестве структурных моделей, как и при проектировании аппаратуры, применяются такие модели, как автомат Мура, автомат Мили, смешанный автомат;

– каждый автомат не должен зависеть от глубокой предыстории ("будущее зависит от настоящего и не зависит от прошлого"), что достигается, если не использовать флаги и умолчания;

– для описания поведения автоматов применяется такой визуальный формализм, как графы переходов;

– каждый граф переходов должен быть семантически и синтаксически корректен. Первое свойство определяет, правильная ли построена модель, а второе - правильно ли она построена. При проверке синтаксической корректности граф переходов проверяется на достижимость, полноту, непротиворечивость, реализуемость и отсутствие генерирующих контуров;

– число вершин в графе переходов должно совпадать с числом состояний в автомате, что наиболее характерно для автоматов Мура без флагов и умолчаний;

– в общем случае при алгоритмизации применяется не один автомат, а система взаимосвязанных автоматов, которая описывается системой взаимосвязанных графов переходов, что поддерживает возможность композиции и декомпозиции алгоритмов и обеспечивает практическое применение технологии при построении сложных систем управления;

– в программирование введено новое понятие "кодирование состояний", широко используемое при проектировании аппаратуры;

– в качестве основного вида кодирования применяется многозначное кодирование, позволяющее использовать только одну переменную для различения произвольного числа

состояний, что особенно важно в случае, когда в разных состояниях формируются одинаковые значения выходных воздействий;

– значение многозначной переменной состояния автомата полностью характеризует "положение" программы, реализующей один автомат, в пространстве ее состояний. Это позволяет ввести в программирование понятие "наблюдаемость" (по одной внутренней переменной);

– для обеспечения наблюдаемости каждого состояния автомата на программную реализацию накладывается ограничение, состоящее в том, что в каждом программном цикле в автомате выполняется не более одного перехода;

– при программной реализации N автоматов с любым числом состояний даже с учетом их взаимодействия используется только N многозначных переменных, кодирующих состояния этих автоматов;

– "наблюдаемость" каждого автомата обеспечивает возможность одновременной индикации значений переменных состояний всех автоматов на одном экране;

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

– указанная реализация графа переходов эквивалентна схеме алгоритма, начинающейся с дешифратора состояний, а не с дешифратора входных воздействий, как это делается традиционно;

– при формальном и изоморфном переходе к программе от графа переходов без флагов и умолчаний последний является сертификационным тестом для ее проверки;

– кроме обмена номерами состояний между автоматами, реализуемыми последовательным расположением в программе конструкций switch, графы переходов могут взаимодействовать и по принципу вложенности, что может быть реализовано вложенными конструкциями switch произвольной глубины или вызовом функций, построенных из этих конструкций, которые реализуют графы переходов соответствующего уровня вложенности;

– графы переходов могут использоваться также и при создании моделей объектов логического управления. Это позволяет с единых позиций проводить описание и выполнять моделирование как разомкнутого, так и замкнутого комплекса "управляющий алгоритм - модель объекта управления";

– предлагаемая технология позволяет Заказчику, Проектанту (Технологу), Разработчику, Программисту, Пользователю и Контролеру однозначно понимать, что должно быть сделано, что делается и что сделано в программно реализуемом проекте.

"Реактивные" системы обладают спецификой, требующей разработки варианта SWITCH-технологии. Перечислим основные отличия рассматриваемых классов систем.

Если в системах логического управления в качестве входных воздействий используются опрашиваемые программой двоичные входные переменные и предикаты, соответствующие определенным состояниям автоматов, взаимодействующих с рассматриваемым автоматом, то для "реактивных" систем это понятие расширено. Во-первых, в качестве входных переменных используются любые подпрограммы (функции),

возвращающие двоичные значения, а во-вторых, введены события, не только обеспечивающие возможность выполнения переходов в автоматах, но инициирующие их запуск. События могут также инициировать реализацию выходных воздействий в случае, когда состояние автомата не изменяется.

Другое отличие "реактивных" систем от систем логического управления состоит в том, что в них в качестве выходных воздействий применяются не двоичные переменные, а произвольные подпрограммы.

Еще одно отличие систем рассматриваемого класса по сравнению с системами логического управления состоит в используемой операционной системе. Если в системах логического управления при их реализации на программируемых логических контроллерах применяются "встроенные" операционные системы, работающие обычно циклически, то для "реактивных" систем характерно применение операционных систем реального времени (ОС РВ), таких как, например, ОС РВ QNX, свойства которых влияют на программную реализацию алгоритмов. В рассматриваемых системах прерывания обрабатываются операционной системой и передаются программе в виде сообщений, которые, в свои очередь, обрабатываются как события с помощью соответствующих обработчиков. При этом после обработки очередного события автомат сохраняет свое состояние и "засыпает" до появления следующего события.

Также как и в системах логического управления, в "реактивных" системах алгоритмы представляются в виде системы взаимосвязанных автоматов. При этом, если в системах первого типа взаимодействие автоматов в основном осуществляется за счет обмена номерами состояний, а вложенность присутствует в "зачаточном" состоянии, то в "реактивных" системах число способов взаимодействия увеличилось.

Кроме того, если в системах логического управления наиболее целесообразно применять такую структурную модель как автомат Мура, то в "реактивных" (событийных) системах более рационально использовать другую модель — смешанный автомат, совмещающий в себе свойства автоматов Мура и Мили.

Указанные особенности событийных систем сделали целесообразным:

– их разработку в виде системонезависимой и системозависимой частей;

– разработку новой структурной схемы систем этого класса, в которой повышен уровень централизации управления за счет выноса логики из обработчиков событий с целью формирования системы взаимосвязанных автоматов, которые запускаются из обработчиков событий;

– реализацию функций входных и выходных воздействий отдельно от автоматов и вызов этих функций из функций, соответствующих автоматам;

– разработку схемы взаимодействия автоматов;

– изменение нотации, используемой при построении графов переходов, особенно в части отображения вложенных автоматов;

– разработку шаблона для формальной и изоморфной реализации графа переходов, в вершины которого могут быть вложены автоматы;

– осуществление взаимодействия автоматов в системе за счет обмена номерами состояний, вложенности (в состояния) и вызываемости (из выходных воздействий);

– автоматическое ведение протокола, в котором при фиксированных значениях входных переменных, указываются значения этих переменных, события, запуск автоматов, их состояния в момент запуска, переходы в новые состояния,

завершение работы автоматов, выходные воздействия и время начала выполнения каждого из них;

– автоматическое построение "короткого" протокола в дополнение к "полному", в котором фиксируются только события и инициируемые ими выходные воздействия, интересующие Заказчика.

Разрабатываемая технология поддерживает все этапы создания программного обеспечения: изучение предметной области, анализ, проектирование, реализация, отладка, сертификация и документирование.

Технология, описываемая в настоящей работе, применялась авторами при создании программного обеспечения для ряда "реактивных" систем, что отражено в соответствующих разделах отчета.

1. Классификация задач, решаемых с использованием автоматного подхода

Опыт разработки и применения автоматного программирования (SWITCH-технология) позволил выделить три класса задач, в которых может быть применен предлагаемый подход:

– системы логического управления, характерной особенностью которых является двоичная форма представления входных и выходных воздействий и ввод входных воздействий путем опроса;

– "реактивные" системы, характерной особенностью которых является наличие среди входных воздействий событий;

– вычислительные алгоритмы, такие как, например, алгоритмы сортировки, поиска и т.п.

Специфика каждого из перечисленных классов задач является причиной различных особенностей в реализации (табл. 1).

Таблица 1. Особенности реализации различных классов задач, решаемых с использованием автоматного подхода

Особенности реализации	Классы решаемых задач		
	Системы логического управления	"Реактивные" (событийные) системы	Вычислительные алгоритмы
Типы входных воздействий автомата (без учета обмена номерами состояний)	• двоичные переменные	• двоичные переменные • события	• двоичные переменные
Типы выходных воздействий автомата	• действия • деятельности	• действия	• действия
Способы реализации входных и выходных воздействий	• переменные • функции	• переменные • функции	• функции
Способы запуска автоматов	• бесконечный цикл • периодический	• по событиям	• конечный цикл

Различные варианты реализации перечисленных классов задач порождают различные структуры получаемого в результате программного обеспечения.

Важнейшая особенность структур программного обеспечения, разрабатываемого с использованием SWITCH-технологии, заключается в том, что в центре находится управляющий автомат, а в общем случае — система взаимосвязанных автоматов [1-3]. При этом структуры программного обеспечения можно классифицировать по способу реализации входных и выходных воздействий и способу запуска автоматов (рис. 1).

Рис. 1. Классификация способов реализации программ, разрабатываемых с применением SWITCH-технологии

Отметим, что каждый из указанных способов запуска выполняется циклически, а их основные отличия состоят в особенностях реализации тела цикла.

Рассмотрим основные из множества возможных структур программного обеспечения, получаемых путем комбинации различных способов реализации.

1.1. Системы логического управления

1.1.1. Реализация на программируемых логических контроллерах

В системах логического управления (при их реализации на программируемых логических контроллерах) автоматы запускаются в бесконечном цикле в соответствии с принятой в контроллерах организацией исполнения управляющих программ. При этом тело цикла состоит из трех частей (рис. 2).

Рис. 2. Структура программного обеспечения систем логического управления, реализуемых на программируемых логических контроллерах

Сначала контроллер осуществляет опрос входных переменных и помещает их текущие значения в оперативную память (выполняется фотография входов). После этого все автоматы выполняются последовательно в заданном порядке. В завершении цикла контроллер записывает текущие значения выходных переменных в выходные регистры, после чего цикл повторяется. Особенность данного способа заключается в том, что шаги цикла выполняются с частотой, определяемой производительностью контроллера. Благодаря этому задержка реакции управляющей программы на изменение значений входных переменных не будет превышать длительности одного программного цикла.

Таким образом, рассмотренная реализация соответствует пунктам 1.1 и 2.1.1 классификации (рис. 1).

1.1.2. Моделирование контроллера в однозадачных операционных системах

Используемый в контроллерах способ выполнения управляющих программ может быть промоделирован на компьютере. Для реализации бесконечного цикла в случае применения языка Си может использоваться оператор for(;;), а выходной преобразователь в общем случае представляет собой набор операторов условного перехода, выбирающих необходимые выходные воздействия в зависимости от значений выходных переменных.

При реализации систем управления на компьютерах, входные и выходные воздействия могут быть реализованы в виде функций (рис. 3). Благодаря этому появляется возможность создания с использованием SWITCH-технологии систем управления более широкого класса, чем системы логического управления. Такая реализация соответствует пунктам 1.2 и 2.1.1 классификации (рис. 1).

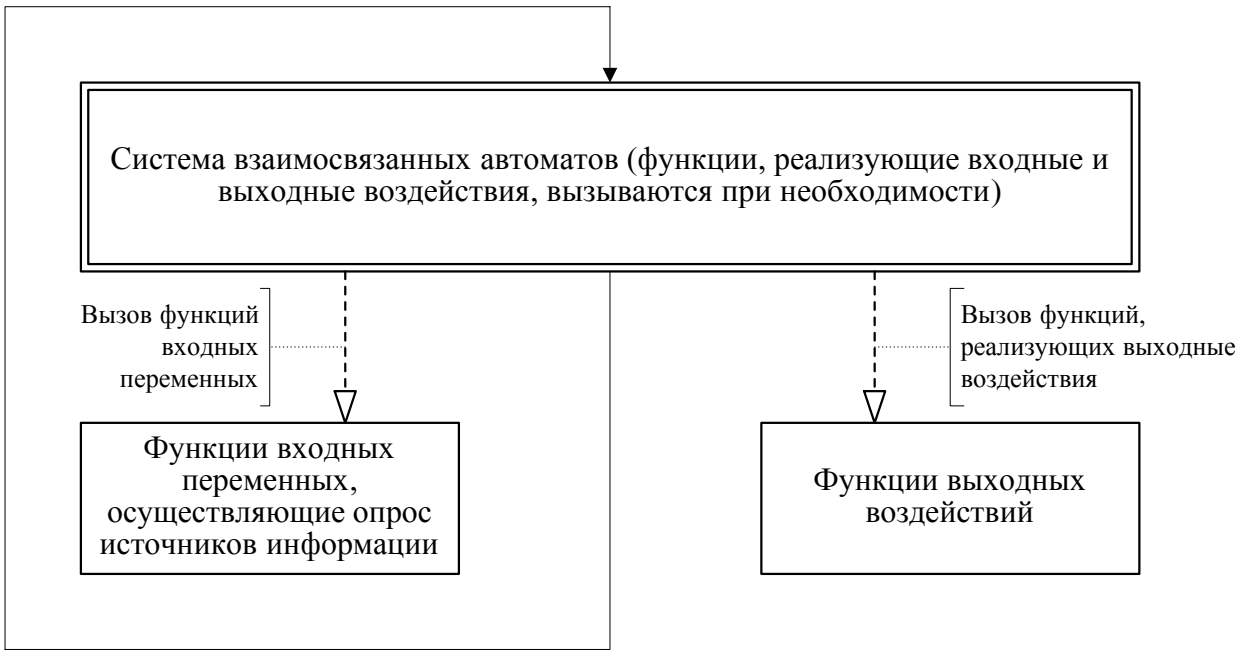

Рис. 3. Структура программного обеспечения систем управления с реализацией входных и выходных воздействий в виде функций

Реализация входных и выходных воздействий в виде функций имеет существенные отличия по сравнению с их реализацией в виде переменных. Из структуры программного обеспечения исчезают входной и выходной преобразователи, так как функции, реализующие входные переменные вызываются только тогда, когда эти переменные встречаются в условиях переходов, а функции, реализующие выходные воздействия, вызываются однократно при выдаче соответствующего воздействия. При этом количество выходных функций удваивается по сравнению с аналогичным количеством выходных переменных. Например, для управления световым индикатором, который находится в выключенном или включенном состоянии, может использоваться одна выходная переменная, нулевое значение которой выключает индикатор, а значение равное единице — включает. При реализации выходных воздействий в виде функций, для управления таким индикатором потребуются две выходные функции, одна из которых выключает индикатор, а другая включает. При желании две эти функции могут быть заменены одной функцией с параметром и условным переходом внутри нее.

1.1.3. Моделирование контроллера в многозадачных операционных системах

Рассмотренный выше способ запуска автоматов в непрерывном цикле является естественным для контроллеров, может быть использован в программном обеспечении, функционирующем под управлением однозадачных операционных систем, но является нежелательным в случае использования многозадачных операционных систем.

При разработке программ для многозадачных операционных систем, непрерывный цикл может быть заменен циклом, тело которого выполняется только при приходе какого-либо сообщения, например сообщения о срабатывании

таймера, период срабатывания которого равен требуемому времени задержки реакции системы управления на изменение значений входных переменных.

Сообщения от таймера принадлежат к разновидности входных воздействий, называемых событиями. Отметим, что в большинстве контроллеров отсутствует возможность порождения и обработки программных событий.

При запуске автоматов от таймера-генератора синхроимпульсов в структуру программы добавляется обработчик события "Срабатывание таймера" (рис. 4). Такая реализация соответствует пунктам 1.2 и 2.2 классификации (рис. 1).

Рис. 4. Структура программ с периодическим запуском автоматов от таймера

При запуске автоматов от таймера из структуры программы исчезает в явном виде бесконечный цикл. Однако

запуск автоматов остается циклическим, учитывая циклический способ обработки событий в многозадачных операционных системах (рис. 5).

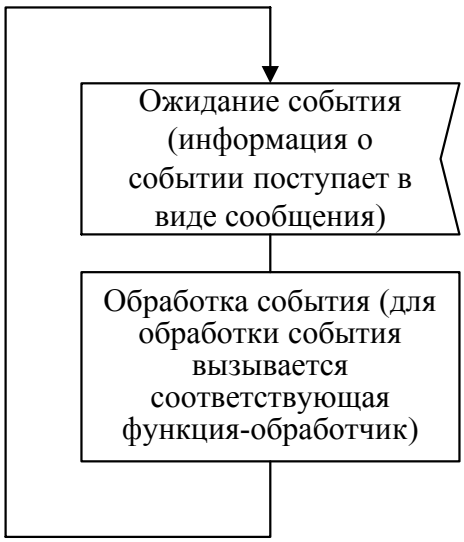

Рис. 5. Обработка событий
в многозадачных
операционных системах

Для уменьшения загрузки процессора в многозадачной системе период срабатывания таймера, запускающего автоматы, может изменяться в зависимости от состояния, в котором находится система управления.

Пусть, например, к разрабатываемой системе управления, функционирующей в режимах "Останов", "Работа" и "Авария" предъявлено требование, состоящее в том, что задержка реакции на изменение входных воздействий не может превышать 0.1 с. Для выполнения этого требования необходимо запрограммировать таймер, используемый в качестве генератора синхроимпульсов, на период, не превышающий указанный. Однако далее это требование может быть уточнено и выяснится, например, что оно предъявлялось только для режима "Работа", а в режиме "Авария" задержка реакции ограничивается значением 1 с, в режиме "Останов" — 10 с.

1.2. "Реактивные" (событийные) системы

В отличие от систем логического управления, в событийных системах входные воздействия разделяются на два типа: входные переменные и события. Возможность применения в качестве входных воздействий событий позволяет использовать SWITCH-технологию при создании программного обеспечения систем управления, принадлежащих к "реактивным" системам (системам, реагирующим на события). К таким системам относятся, например, все системы реализующие пользовательский интерфейс в современных графических оболочках.

При этом развивая структуру программ, представленную на рис. 4, в которой автоматы запускались из обработчика события срабатывания таймера, будем запускать автоматы из обработчиков всех событий, присутствующих среди входных воздействий.

Отметим, что один и тот же источник входных воздействий может одновременно являться источником и входных переменных и событий. Поясним это на примере кнопки, которая может являться источником входных воздействий (рис. 6), перечисленных ниже:

– событие "Нажатие кнопки";

– событие "Отпускание кнопки";

– входная переменная "Кнопка нажата" (либо ее инверсия — "Кнопка отпущена").

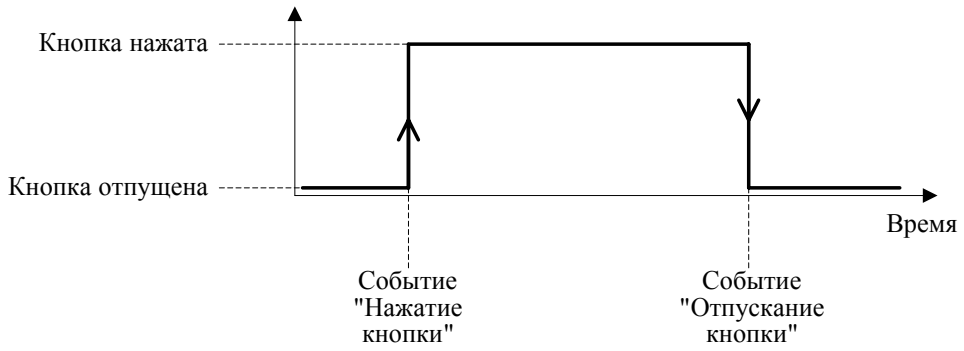

Рис. 6. Временная диаграмма нажатия кнопки

Таким образом, входные переменные указывают на текущее значение входного воздействия, а события — на изменение этого значения. Поэтому входные переменные на временной диаграмме отмечаются на вертикальной оси, а события — на оси времени. Такое время в [4] названо "событийным".

Из изложенного следует, что в событийных системах время присутствует в явном виде среди входных воздействий, в отличие от систем логического управления, в которых оно может присутствовать только в выходных воздействиях (функциональные элементы задержки). В вычислительных алгоритмах время в явном виде не используется.

В качестве названий двоичных входных переменных используются предложения в повествовательной форме (например, "Кнопка нажата"). Названия событий содержат отглагольные существительные (например, "Нажатие кнопки").

В случае, когда необходимо решить, в каком виде рассматривать входное воздействие, возможны варианты, перечисленные ниже.

1. Входная переменная. Этот вариант является наиболее предпочтительным, не считая ситуаций, описанных в следующих пунктах.

2. Событие. Использование событий целесообразно, в первую очередь, для входных воздействий имеющих кратковременный характер. К источникам таких воздействий можно отнести, например, кнопки без памяти и таймеры. Отметим однако, что факт возникновения подобных входных воздействий может быть запомнен, после чего они могут рассматриваться как входные переменные.

3. Входная переменная и событие одновременно. Для ускорения реакции на изменение входной переменной, это изменение может рассматриваться как событие. В качестве примера входного воздействия, которое необходимо рассматривать одновременно и как входную переменную, и как событие, можно привести сигнализатор открытого положения двери микроволновой печи. Входное воздействие от этого сигнализатора должно рассматриваться как входная переменная, так как его необходимо опрашивать, проверяя допустимость включения печи. В то же время, при открытии двери печи во время ее работы, она должна быть отключена немедленно, а не тогда, когда входная переменная "Дверь открыта", будет опрошена в результате прихода какого-либо другого события. Отметим, что в данном примере рассматривается модель микроволновой печи, так как в реальности открывание двери непосредственно размыкает ее питающую цепь.

Запуск автоматов в событийных системах происходит из обработчиков событий. При этом в функцию, реализующую автомат, номер произошедшего события передается в виде параметра. Структура программ в общем случае содержит (рис. 7):

– обработчики событий;

– систему взаимосвязанных автоматов;

– функции, реализующие входные переменные;

– функции, реализующие выходные воздействия;

– функции протоколирования;

– вспомогательные модули.

Такая реализация соответствует пунктам 1.2 и 2.3 классификации (рис. 1).

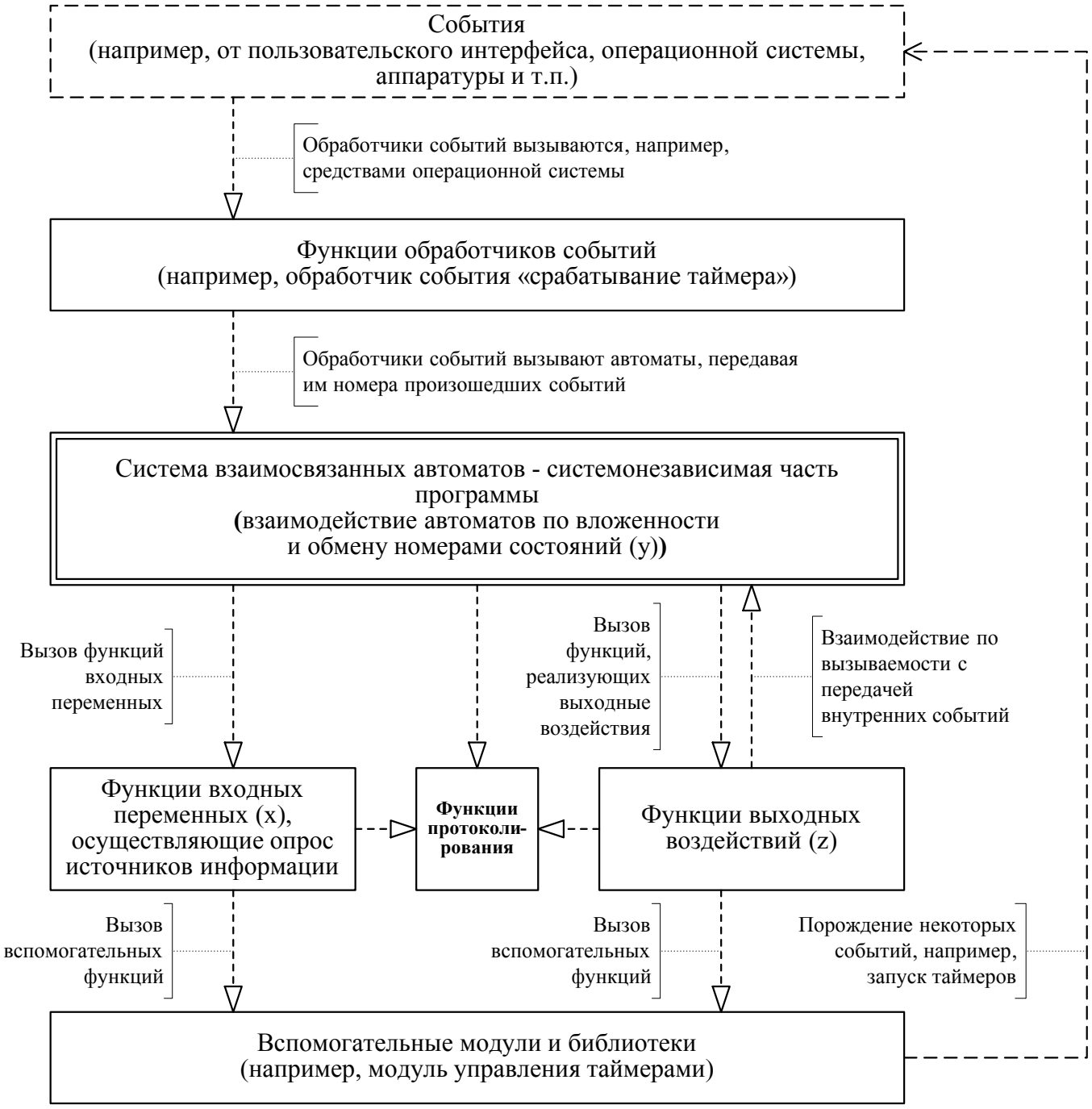

Рис. 7. Структура программного обеспечения реактивных систем

2. Автоматы

В схемах связей, графах переходов и схемах взаимодействия автоматов могут использоваться обозначения, приведенные в табл. 2.

Таблица 2. Сокращенные обозначения, применяемые в схемах связей, графах переходов и схемах взаимодействия автоматов

Обозначение	Расшифровка
An	Автомат с номером n
yn	Переменная состояния автомата с номером n
yb = p	Предикат, проверяющий равенство номера текущего состояния автомата с номером b значению p. При программной реализации является сокращенной записью предиката "yb == p"
xj	Входная переменная с номером j
Cn_r	Произвольная логическая формула с номером r для автомата с номером n
en	Событие с номером 'n'. При программной реализации обозначение "en" является сокращенной записью предиката "e == n", а "$\overline{\text{en}}$" — сокращенной записью предиката "e != n"
zk	Выходное воздействие с номером 'k'
zd_m	'm'-ое значение выходного воздействия с номером 'd'
Ts	Таймер с номером 's'
i:	Численное обозначение приоритета условия перехода ('1' - наивысший приоритет)

2.1. Нотация схемы связей

При построении схемы связей автомата используется нотация, приведенная на рис. 8.

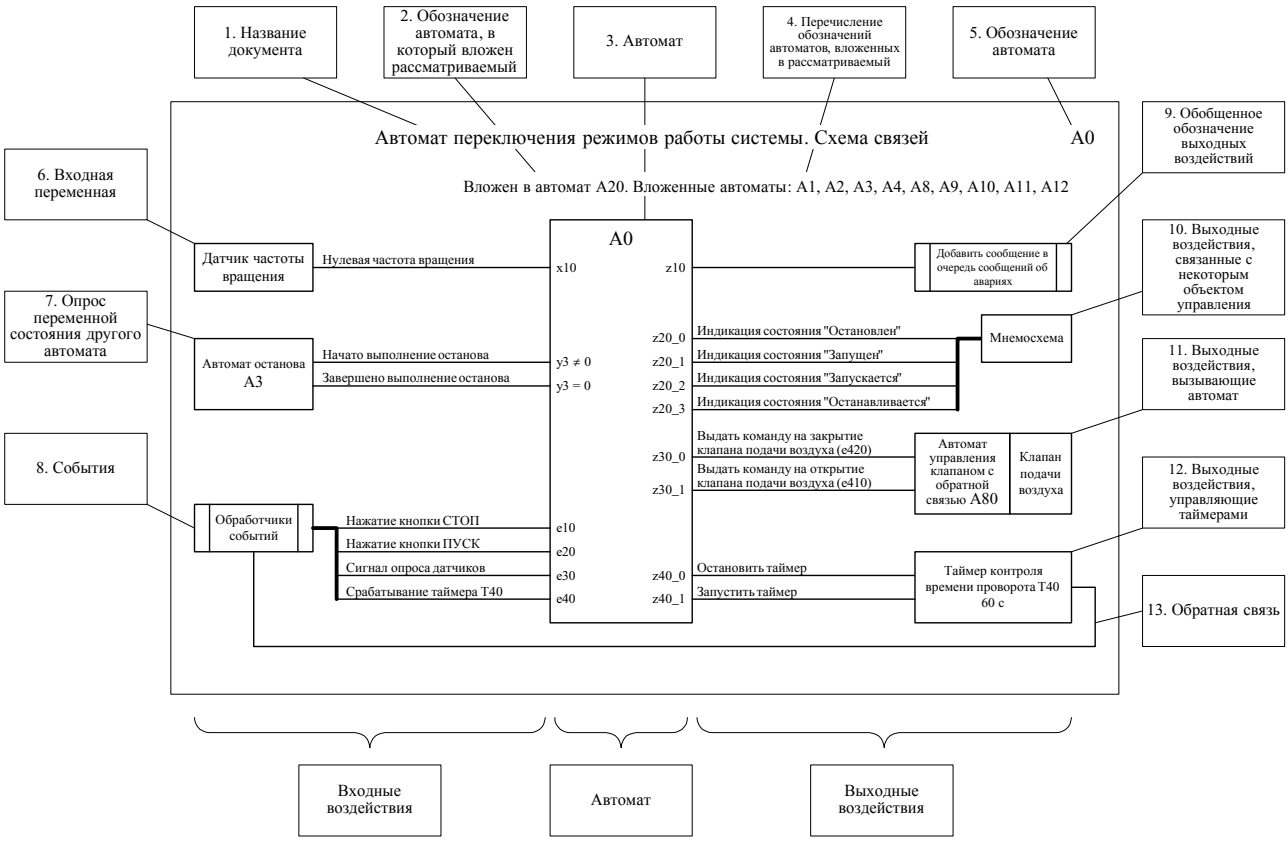

Рис. 8. Нотация, используемая в схемах связей автоматов

В рассматриваемой нотации используются перечисленные ниже обозначения.

1. Название документа. Название документа располагается посередине в верхней части листа и содержит название автомата и название собственно документа — "Схема связей".

2. Обозначение автомата, в который вложен рассматриваемый. Содержит информацию о том, в какой автомат вложен рассматриваемый автомат. Если автомат не является вложенным, то данный элемент нотации отсутствует.

3. Автомат. На схеме связей автомат условно обозначается прямоугольником, в верху которого приведено обозначение автомата. Ширина прямоугольника выбирается таким образом, чтобы внутри него свободно помещались обозначения входных и выходных воздействий.

4. Перечисление обозначений автоматов, вложенных в рассматриваемый. Если рассматриваемый автомат не содержит вложенных автоматов, то данный элемент нотации отсутствует.

5. Обозначение автомата, располагается в правом верхнем углу листа на одном уровне с названием документа.

6. Входная переменная. Перечисление входных переменных выполняется в левой верхней части схемы связей. Для каждой входной переменной указывается ее источник, соединенный горизонтальной линией с прямоугольником, обозначающим автомат. Над этой линией записывается полное название входной переменной, а внутри прямоугольника, обозначающего автомат, в месте его соединения с линией — обозначение входной переменной.

7. Опрос переменной состояния другого автомата. Если в условиях переходов автомата присутствует предикат, проверяющей значение переменной состояния другого автомата, то на схеме связей это отображается по аналогии с входной переменной, после перечисления входных переменных. Указывается обозначение и название автомата, переменная состояния которого опрашивается, над соединительной линией указывается смысловое значение предиката, а внутри обозначающего автомат прямоугольника — сам предикат.

8. События. Обработчики событий указываются обобщенно в левой нижней части схемы связей. Для каждого события рисуется линия, соединяющая условное обозначение

обработчиков событий с условным обозначением автомата. Над этой линией записывается название события, а внутри прямоугольника, обозначающего автомат, в месте его соединения с линией — обозначение события.

9. Обобщенное обозначение выходных воздействий. В общем случае выходные воздействия обозначаются с использованием символа "Подпрограмма", внутри которого записывается название этого выходного воздействия. Этот символ соединяется линией с прямоугольником, обозначающим автомат, внутри которого записывается обозначение выходного воздействия.

10. Выходные воздействия, связанные с некоторым объектом управления. Если связь выходного воздействия с некоторым объектом управления может быть легко проиллюстрирована, то записывается название этого объекта, а полное название выходного воздействия переносится на соединительную линию.

11. Выходные воздействия, вызывающие автомат. Если из выходного воздействия осуществляется вызов автомата, то указываются обозначение и название этого автомата, а после названия выходного воздействия указывается событие, с которым вызывается автомат. Как следствие изложенного отметим, что автоматы могут вызываться не только из состояний, но также с дуг и петель.

12. Выходные воздействия, управляющие таймерами. Для выходных воздействий, управляющих таймерами указываются обозначение и название таймера. При этом номера таймера, выходных воздействий, управляющих им, и порождаемого им события должны совпадать.

13. Обратная связь. Во всех случаях, когда это позволяет размер схемы связей, на ней должны быть указаны обратные связи от объектов управления ко входам автомата.

2.2. Нотация графов переходов

При построении графов переходов автомата используется нотация, содержащая приведенные на рис. 9 обозначения.

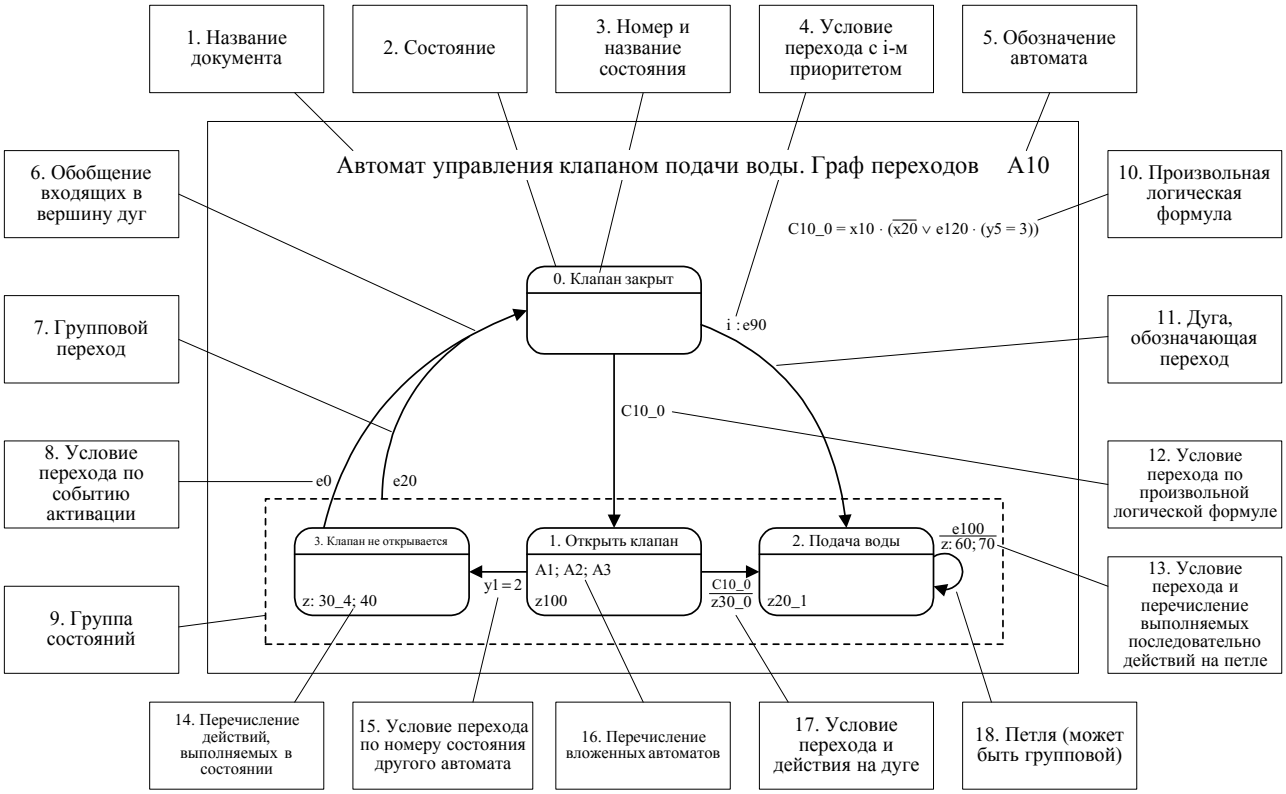

Рис. 9. Нотация, применяемая в графах переходов автоматов

В рассматриваемой нотации используются перечисленные ниже обозначения.

1. Название документа. Название документа располагается посередине в верхней части листа и содержит полное название автомата и название собственно документа — "Граф переходов".

2. Состояние. Состояние автомата (вершина графа переходов) обозначается прямоугольником с закругленными углами с типовым соотношением ширины к высоте 2:1, разделенным на три горизонтальные области одинаковой высоты.

3. Номер и название состояния. Верхняя область вершины графа переходов содержит номер состояния автомата и, через точку, краткое название этого состояния.

4. Условие перехода с i-м приоритетом. Условия переходов могут предваряться номером приоритета этого перехода, отделенного двоеточием. Условия переходов записываются ближе к началу дуги, которую помечают, таким образом, чтобы исключить неоднозначное толкование принадлежности к той или иной дуге.

5. Обозначение автомата, располагается в верхнем правом углу листа на одном уровне с названием документа.

6. Обобщение входящих в вершину дуг. Входящие в вершину дуги могут быть объединены в единую линию для более компактного их отображения.

7. Групповой переход. Переход, общий для группы состояний, называется групповым. Петли также могут быть групповыми.

8. Условие перехода по событию активации. Событие активации, получаемое автоматом от своего головного автомата имеет сокращенное название e0.

9. Группа состояний. Состояния, имеющие общие переходы, могут быть объединены в группы для более компактного отображения этих переходов.

10. Произвольная логическая формула. Если некоторая логическая формула повторяется в условиях переходов автомата несколько раз или слишком длинна, она может быть записана вне графа переходов на том же листе и обозначена идентификатором, начинающимся с буквы 'C', после которой следует номер автомата и порядковый номер формулы.

11. Дуга, обозначающая переход. Переходы автомата обозначаются в графе переходов дугами, соединяющими соответствующие вершины.

12. Условие перехода по произвольной логической формуле. Идентификаторы, обозначающие вынесенные из автомата произвольные логические формулы, используются в условиях переходов также, как и другие входные воздействия.

13. Условие перехода и перечисление выполняемых последовательно действий на петле. Петли, как и дуги, могут помечаться приоритетами, условиями перехода и действиями, выполняемыми при переходе. Так как при переходе по петле автомат сохраняет свое состояние, то петли используются для обозначения действий, выполняемых в состоянии при некоторых условиях.

14. Перечисление действий, выполняемых в состоянии. Нижняя область вершины графа переходов может содержать перечисление выходных воздействий, выполняемых в соответствующем состоянии. В случае, если в состоянии выполняется одно единственное выходное воздействие, записывается его сокращенное название (например, "z100"). В случае, когда в состоянии выполняется несколько выходных воздействий, символ 'z' записывается только один раз, а после него ставится двоеточие и перечисляются разделенные точкой с запятой номера выходных воздействий (например, "z: 30_4; 40").

15. Условие перехода по номеру состояния другого автомата. В условиях переходов автомата может присутствовать предикат, проверяющий значение переменной состояния другого автомата.

16. Перечисление вложенных автоматов. Средняя область вершины графа переходов может содержать перечисление автоматов, вложенных в это состояние. Если в состояние вложено более одного автомата, то при перечислении

сокращенные названия автоматов разделяются точкой с запятой.

17. Условие перехода и действия на дуге. Кроме приоритетов и собственно условий переходов дуги могут также помечаться перечислением выходных воздействий, выполняемых при переходе. В этом случае выходные воздействия записываются под условием перехода и отделяются от него горизонтальной линией. Выходные воздействия перечисляются также, как и в вершинах графа переходов.

18. Петля. Сохранение автоматом своего состояния на графе переходов обозначается петлями, направленными по часовой стрелке. Состояния автомата предполагаются устойчивыми, поэтому петли, не содержащие выходных воздействий, умалчиваются. Петли могут быть групповыми.

На рис. 10 нотация, применяемая в графах переходов при описании состояний, рассматривается более подробно.

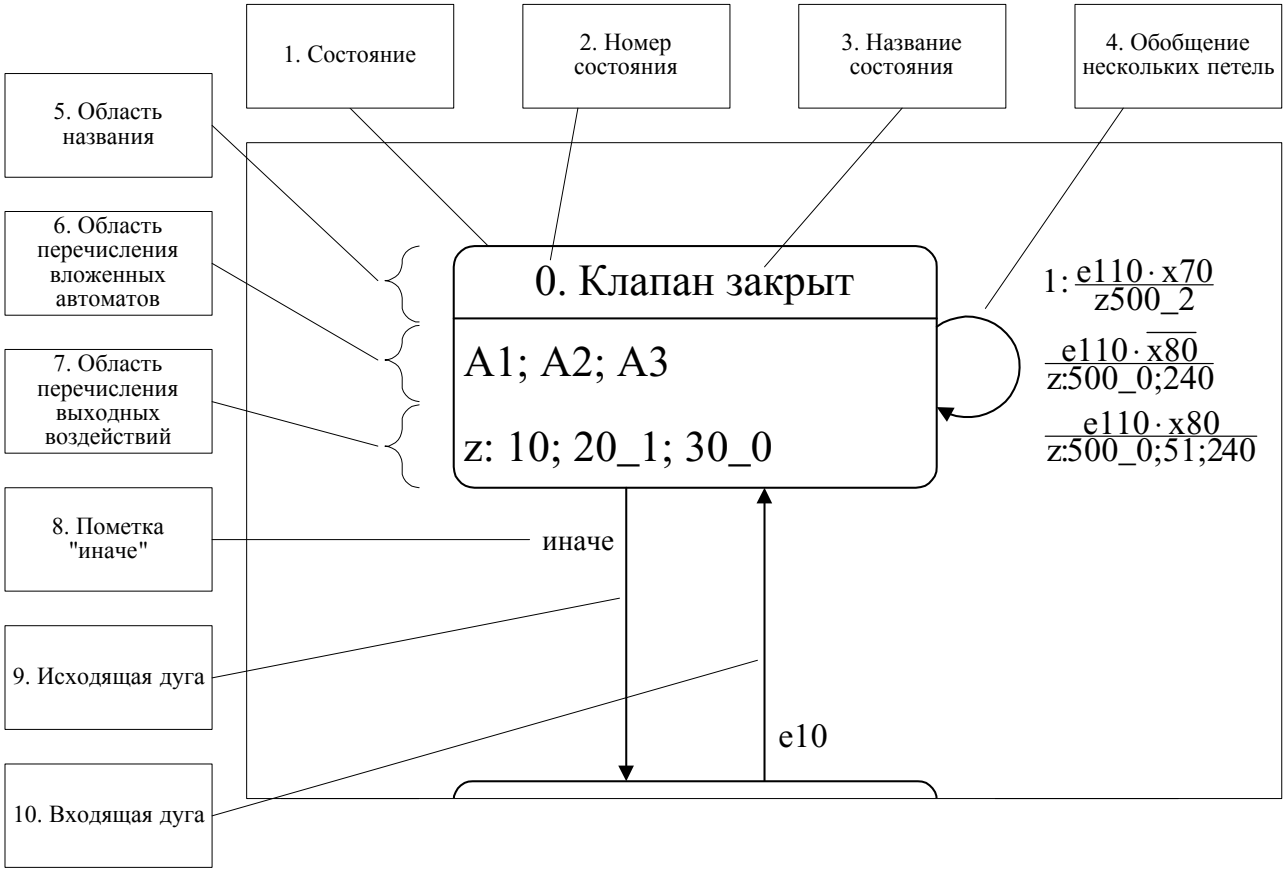

Рис. 10. Нотация, используемая в графах переходов автоматов при описании вершин

В рассматриваемой нотации используются перечисленные ниже обозначения.

1. Состояние. При необходимости высота прямоугольника, обозначающего вершину графа переходов может быть увеличена.

2. Номер состояния. В качестве номеров состояний применяются целые числа. При этом должна обеспечиваться уникальность номеров внутри одного графа переходов. Нумерация (кодирование) состояний начинается с нуля (нулевой номер присваивается начальному состоянию) и далее, по возможности без пропусков. Для отображения номера состояния используется тот же шрифт, что и для отображения названия состояния.

3. Название состояния. Название состояния может определяться состоянием объекта управления и, по возможности, должно быть кратким.

4. Обобщение нескольких петель. Если у вершины графа переходов имеется несколько петель, пометки этих петель могут быть записаны обобщенно для большей компактности.

5. Область названия. Краткое название состояния записывается после номера состояния с использованием пропорционального шрифта. При необходимости размер используемого шрифта может меняться. Если применяется слишком длинные названия состояний, то они могут быть записаны в несколько строк, при этом высота области названия соответствующим образом увеличивается.

6. Область перечисления вложенных автоматов. Обозначения вложенных автоматов перечисляются через точку с запятой с использованием пропорционального шрифта. При необходимости размер используемого шрифта может меняться, а высота области может увеличиваться.

7. Область перечисления выходных воздействий. Выходные воздействия записываются с использованием пропорционального шрифта. При необходимости размер используемого шрифта может меняться, а высота области — увеличиваться.

8. Пометка "иначе". Некоторые дуги на графах переходов могут помечаться словом "иначе". Это означает, что указанный переход выполняется в случае, если не выполнились все остальные условия переходов на всех исходящих дугах и петлях рассматриваемой вершины. Условие "иначе" не может иметь приоритета и не может помечать групповые дуги и петли.

9. Исходящая дуга. Дуга, обозначающая переход из рассматриваемого состояния в какое-либо другое, является для соответствующей вершины исходящей.

10. Входящая дуга. Дуга, обозначающая переход из какого-либо другого состояния в рассматриваемое, является для соответствующей вершины входящей.

2.3. Нотация схемы взаимодействия автоматов

Автоматы могут взаимодействовать между собой за счет обмена номерами состояний, вложенности и вызываемости. Они также могут быть одновременно вложенными и вызываемыми.

Указанные виды взаимодействия формализуются с помощью схемы взаимодействия автоматов, нотация для построения которой приведена на рис. 11.

Рис. 11. Нотация схемы взаимодействия автоматов

1. Название документа. Название документа располагается посередине в верхней части листа и содержит название разрабатываемой системы и название собственно документа — "Схема взаимодействия автоматов".

2. Обобщенное изображение обработчиков событий. В верхней части схемы обобщенно изображаются обработчики событий, вызывающие автоматы.

3. Вызов автомата из обработчиков событий. Вызов автоматов непосредственно из обработчиков событий обозначается пунктирной линией со стрелкой.

4. Автомат. Автоматы на схеме взаимодействия автоматов обозначаются прямоугольниками, содержащими название и обозначение автомата. В примыкающих снизу прямоугольниках, могут указываться перечни автоматов, взаимодействующих с рассматриваемым по обмену номерами состояний и по вызываемости.

5. Вложенность. Вложенность обозначается линией, заканчивающейся ромбом, по аналогии с обозначением вложенности на диаграммах классов [5].

6. Обозначение автомата. Обозначение автомата указывается в правом верхнем углу соответствующего прямоугольника.

7. Перечень автоматов, переменные состояния которых опрашиваются из рассматриваемого автомата, указывается в прямоугольнике сразу после прямоугольника, его обозначающего.

8. Перечень автоматов, которые опрашивают переменную состояния рассматриваемого автомата, указывается в следующем прямоугольнике.

9. Перечень вызываемых автоматов. Для отражения взаимодействия автоматов по вызываемости, в третьем из

прямоугольников перечисляются автоматы, вызываемые из рассматриваемого.

2.4. Модель автомата

2.4.1. Запуск автомата

Исходя из того, что автомат реализуется функцией (подпрограммой), под его запуском будем понимать вызов этой функции. Из этого определения следует, что переходы и выдача выходных воздействий в автомате происходят только при выполнении реализующей его функции.

Для обеспечения реакции автомата на входные воздействия будем запускать его при возникновении любого события, используемого в автомате. Запуск автомата при поступлении события **e** будем называть "запуск автомата с событием **e**" (например, "запуск автомата с событием нажатия левой кнопки мыши e10").

События, используемые в автомате, будем также называть "события, с которыми запускается автомат". Отметим, что на схеме связей автомата указываются все события, с которыми он запускается, вне зависимости от того, присутствуют ли они в графе переходов этого автомата.

Каждому событию присваивается уникальный номер. При этом номер события, с которым запускается автомат, передается в качестве параметра функции, реализующей автомат. В качестве параметров может передаваться также и дополнительная информация о событии.

Вызов реализующей автомат функции выполняется из обработчиков событий, с которыми запускается автомат. По аналогии с логическим управлением при каждом запуске автомат выполняет не более одного перехода, после чего

реализующая его функция завершает свою работу — возвращает управление в вызвавший ее обработчик события.

Из изложенного следует, что события обрабатываются автоматом последовательно в порядке их поступления. При этом в некоторых реализациях может возникнуть необходимость создания очереди событий для предотвращения их потери.

2.4.2. Входные воздействия и переходы

После запуска автомат выполняет проверку условий переходов из текущего состояния (начальным состоянием считается состояние с номером 0). В первую очередь, проверяются условия переходов на групповых исходящих дугах и групповых петлях в соответствии с выбранными приоритетами. После этого на основании указанных приоритетов проверяются условия переходов на остальных исходящих дугах и петлях. Условия переходов без приоритетов проверяются в произвольном порядке.

При проверке условий переходов определяются значения:

– предикатов, проверяющих номер произошедшего события;

– предикатов, проверяющих номера состояний других автоматов;

– входных переменных.

Если очередное проверяемое условие перехода истинно, дальнейшая проверка прекращается. Автомат выполняет указанные на соответствующей дуге выходные воздействия (при их наличии) и переходит в новое состояние. Если истинно условие перехода, указанное на петле, то выполняются указанные на ней выходные воздействия, а состояние автомата сохраняется.

Входные переменные реализуются функциями (подпрограммами). Опрос входных переменных заключается в

вызове указанных функций из функции, реализующей автомат. После выполнения действий, в общем случае произвольных, реализующая входную переменную функция завершается, возвращая двоичный результат. Благодаря такой реализации опрашиваются только те входные переменные, которые участвуют в проверяемых условиях переходов, в отличие от контроллеров, где опрос всех входных переменных выполняется циклически.

После перехода в новое состояние автомат выполняет указанные в нем выходные воздействия.

2.4.3. Выходные воздействия

Выходные воздействия реализуются функциями (подпрограммами), выполняющими в общем случае произвольные действия. Эти воздействия могут формироваться на дугах, петлях и в вершинах графа переходов.

Выходные воздействия на дуге формируются при истинности соответствующего ей условия перехода, до того, как автомат перейдет в новое состояние. Выходные воздействия на петле формируются аналогично, но без изменения состояния автомата.

Выходные воздействия в вершине формируются после перехода автомата в соответствующее состояние.

2.4.4. Вложенные автоматы

Высказывание "автомат A1 вложен в автомат A0" означает, что автомат A1 вложен в одно или несколько состояний автомата A0. При этом,

– автомат A0 является головным, а автомат A1 вложенным;

– если при запуске автомат A0 находится в состоянии, в которое вложен автомат A1, то последний запускается с тем же событием, с которым был запущен автомат A0;

– при переходе автомата A0 в состояние, в которое вложен автомат A1, последний запускается с событием активации e0.

Продолжая для краткости обозначать головной автомат как A0, а вложенный как A1, рассмотрим различные варианты вложенности.

1. **Вложенность в одно состояние**. Пусть автомат A1 вложен в одно из состояний автомата A0. Это означает, что автомат A1 запускается, во-первых, когда автомат A0 при запуске находится в том состоянии, в которое вложен автомат A1, а во-вторых, при переходе автомата A0 в это состояние. При этом автомат A1 детализирует поведение системы в соответствующем состоянии автомата A0.

2. **Вложенность в несколько состояний**. Пусть автомат A1 вложен в несколько состояний автомата A0. Такой вариант вложенности может рассматриваться как случай взаимодействия параллельных процессов, в котором процесс, определяемый автоматом A1, выполняется только тогда, когда процесс, определяемый автоматом A0, находится в одном из состояний, в которые вложен автомат A1.

3. **Вложенность во все состояния**. Пусть автомат A1 вложен во все состояния автомата A0. Такую суперпозицию можно применять в случае, когда необходимо вызывать вложенный автомат каждый раз, когда головной автомат изменяет свое состояние. При этом, при любом изменении состояния автомата A0, автомат A1 будет вызываться с событием активации e0. Если такая необходимость отсутствует, то это означает, что автомат A1 не взаимодействует по вложенности с автоматом A0 и должен

быть расположен на одном уровне вложенности с автоматом A0.

Отметим, что вложенный автомат A1 может быть преобразован в автомат, расположенный на одном уровне вложенности с автоматом A0. Это достигается соответствующими преобразованиями условий переходов автомата A1, блокирующими какие-либо переходы при определенных состояниях автомата A0. В случае, если все преобразования были выполнены правильно, полученные в результате два автомата будут работать также, как и исходные, но при этом значительно увеличится количество запусков автомата A1.

Обобщим особенности реализации вложенных автоматов.

При запуске автомата с некоторым событием, перед проверкой условий переходов, все вложенные в текущее состояние автоматы последовательно запускаются с тем же событием. В случае многоуровневой вложенности, порядок запуска автоматов определяется их текущими состояниями — путем в схеме взаимодействия автоматов. При этом последовательность запуска и завершения работы автоматов напоминает алгоритм поиска в глубину [6].

Отметим, что при переходе автомата в состояние, содержащее вложенные автоматы, перед формированием выходных воздействий в этом состоянии, вложенные автоматы последовательно вызываются с событием активации e0, которое также как и другие события распространяется вглубь иерархии вложенных автоматов.

3. Создание программного обеспечения событийных систем

3.1. Предлагаемая технология

Предлагаемая технология характеризуется следующими особенностями:

– применяется процедурный (императивный) подход к разработке программ;

– в качестве базового используется понятие "автомат", а не "класс", "объект", "алгоритм" или "агент", как это имеет место при других подходах. В отличие от объектного моделирования [5] построение всех основных моделей основано на применении только автоматной терминологии. При этом 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;

– 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;

– расширена, по сравнению с [1], нотация, применяемая при построении схем связей и графов переходов (рис. 8-11).

3.1.1. Изучение предметной области

На основе технического задания, которое при автоматизации технологических процессов обычно выдается Заказчиком в словесной форме в виде совокупности сценариев и случаев использования [5], строится структурная схема системы, позволяющая получить общее представление об организации управления, применяемой аппаратуре и интерфейсе объекта управления.

3.1.2. Проектирование

Перечислим основные стадии процесса проектирования.

1. Выделяются составные части алгоритма управления, каждая из которых реализуется автоматом (например, автомат управления насосом или автомат контроля температуры). Составляется перечень автоматов.

2. Строится схема взаимодействия автоматов.

3. Состояния каждого автомата первоначально определяются по выделенным состояниям объекта управления в целом или его части, а при большом их количестве — по алгоритму управления, построенному в виде схемы алгоритма [7]. В автоматы также могут быть введены и другие состояния, связанные, например, с неправильными действиями Оператора. Каждый автомат при необходимости декомпозируется. Процесс выделения состояний завершается созданием перечня состояний для каждого автомата.

4. В отличие от традиционного программирования вводится подэтап — кодирование состояний автомата. При этом в каждом автомате его состояниям присваиваются десятичные номера.

5. Связи каждого автомата с его "окружением" формализуются схемой связей автомата, предназначенной для полного описания его интерфейса. В этой схеме указываются 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.

Входные воздействия, которые одновременно являются и событиями и входными переменными, целесообразно рассматривать как входные переменные. Отметим, что некоторые входные переменные могут формироваться в результате сравнения входных аналоговых сигналов с уставками.

6. Строится граф переходов каждого автомата. При этом 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, петли и вершины могут содержать списки последовательно формируемых выходных воздействий.

Вершины в графах переходов, являющиеся устойчивыми, содержат петли. Если на петле не формируются выходные воздействия, то она умалчивается. В противном случае изображаются одна или несколько петель.

Каждый граф переходов проверяется на достижимость, непротиворечивость, полноту и отсутствие генерирующих контуров.

3.1.3. Реализация

На этом этапе строится программа, в которой графы переходов, входные переменные, обработчики событий и выходные воздействия реализуются в виде функций. Кроме того программа содержит вспомогательные модули (например, модуль управления таймерами).

Программа, создаваемая с использованием предлагаемого подхода, состоит из двух частей: системонезависимой и системозависимой.

Система взаимосвязанных автоматов образует системонезависимую (например, от операционной системы) часть программы, которая реализует алгоритм функционирования системы управления.

Обработчики событий, функции входных и выходных переменных и вспомогательные модули образуют системозависимую часть программы.

Каждый граф переходов формально и изоморфно реализуется отдельной функцией на выбранном языке программирования в соответствии с алгоритмом (рис. 12), позволяющим реализовывать произвольные автоматы любого уровня вложенности. Функция, реализующая автомат, строится путем заполнения приведенного ниже шаблона содержащего две конструкции switch и оператор if. Первая конструкция switch запускает 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. Из изложенного следует, что автомат Мили реализуется одним оператором switch, а автомат Мура и смешанный автомат — двумя.

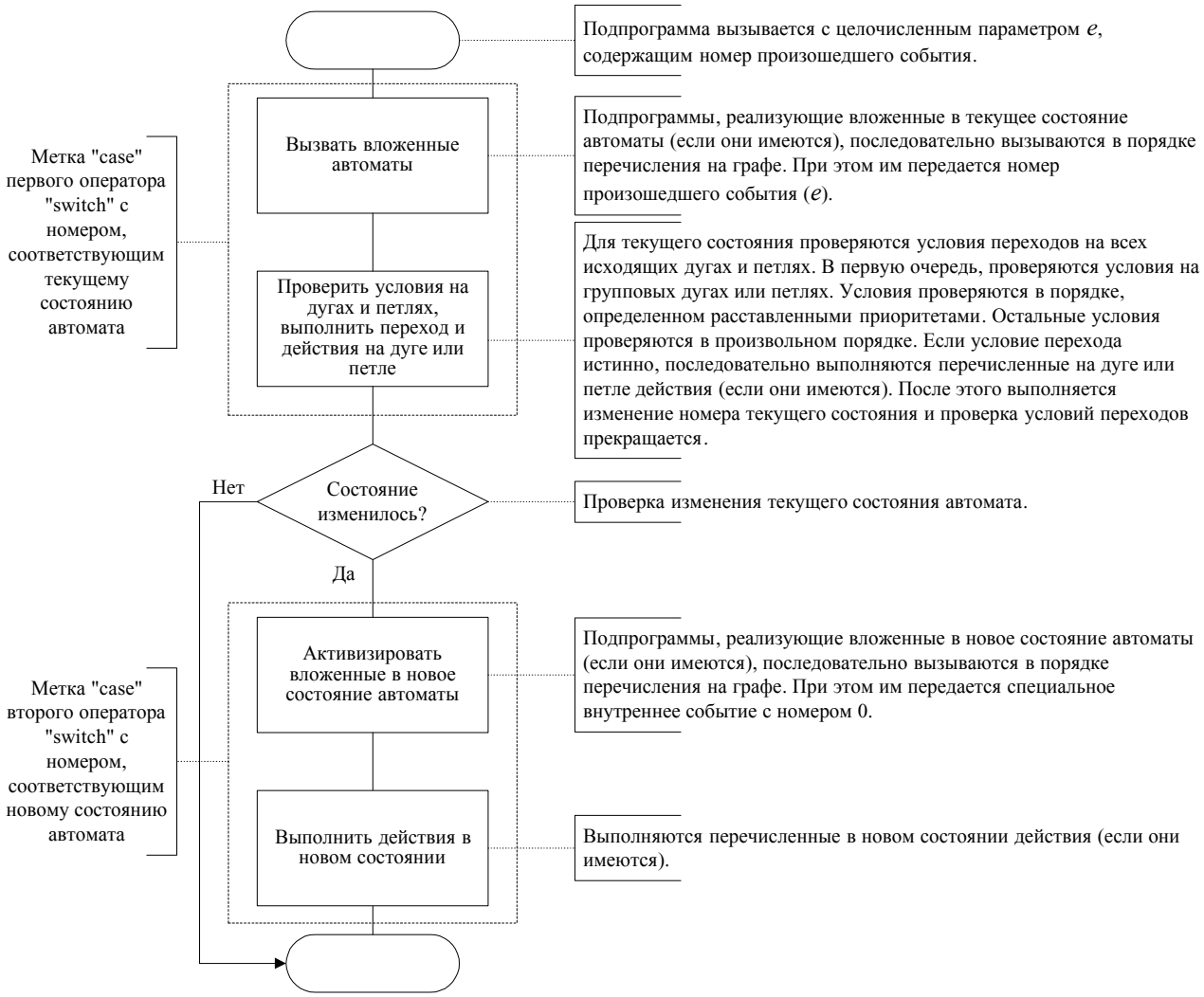

Рис. 12. Алгоритм реализации графов переходов

// Шаблон функции, реализующей автомат. Заменить "_i_" на номер автомата.

```
void A_i_( int e )
{
  int y_old = y_i_ ;

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

  switch( 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,
                   "Ошибка в автомате A_i_: неизвестный номер состояния!", 0 ) ;
      #endif
  } ;

  // Если состояние не изменилось - завершить выполнение функции.
  if( y_old == y_i_ ) goto A_i__end ;

  // Протоколирование перехода в автомате.
  #ifdef A_i__TRANS_LOGGING
    log_trans( "A_i_", y_old ) ;
  #endif

  switch( 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( "A_i_", y_i_, e ) ;
  #endif
} ;
```

После реализации графа переходов текст функции при необходимости должен корректироваться для обеспечения бесповторности опроса входных переменных. Это позволяет решить проблему "риска", заключающуюся в том, что если при проверке условий на исходящих из вершины дугах одна и

та же входная переменная опрашивается более одного раза, то существует возможность, что она изменит свое значение между опросами [1].

Реализация входных переменных, обработчиков событий, выходных воздействий, вспомогательных модулей и пользовательских интерфейсов образует системозависимую часть программы. При этом все функции, реализующие входные переменные, записываются в порядке возрастания их номеров в один файл, а реализующие выходные воздействия — в другой.

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.

Перечислим особенности предлагаемой реализации:

– она позволяет использовать одно и то же выходное воздействие как в вершине, так и на принадлежащей ей петле. При этом при переходе в такую вершину выполняется указанное в ней действие, а при запуске автомата и выполнении условия перехода на этой петле — действие, ее помечающее;

– предназначенные для более компактного представления графов переходов групповые дуги и петли реализуются также как и остальные дуги и петли в графах, повторяясь для каждой из вершин, входящих в группу;

– для хранения номера текущего состояния автомата используется одна внутренняя переменная, а для определения того, что состояние изменилось, применяется вторая переменная, носящая вспомогательный характер;

– 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;

– изоморфизм в реализации графов переходов позволяет при необходимости решить обратную задачу [5]: однозначно восстановить граф переходов по построенной 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 значительным изменениям может подвергаться только системозависимая часть;

– 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 [8] в любой момент времени после начала реализации системозависимой части.

3.1.4. Отладка и сертификация

В рамках предлагаемой технологии отладка программ может выполняться тремя способами: традиционным (интерактивным) способом; наблюдением за значениями переменных состояний всех автоматов, отображаемыми на одном экране; путем анализа протоколов работы программы.

Первый способ включает как простое исполнение программы с различными входными данными, так и использование отладчика. Этот способ отладки является традиционным и, поэтому, подробно не рассматривается.

Второй способ основан на введенном в программирование понятии "наблюдаемость" [1]. При этом для однозначного определения состояний каждого автомата достаточно следить за значениями одной переменной, а для системы взаимосвязанных автоматов — за переменными состояний этих автоматов, которые должны отображаться на одном экране.

Наиболее важным является третий из предлагаемых способов отладки. Рассмотрим его более подробно.

Для обеспечения протоколирования, функции, реализующие автоматы, входные переменные и выходные воздействия, содержат вызовы функций протоколирования. Благодаря этому становится возможным автоматическое получение протоколов работы программы в терминах автоматов. Автоматическое получение таких протоколов доказывает, что графы переходов являются не "картинками", а математическими моделями.

Это чрезвычайно важно, так как "протоколы (история вычислений) являются конструкциями, вскрывающими механизм работы программы и, поэтому, постепенно среди теоретиков программирования сложилось представление, что множество протоколов лучше характеризует программу, нежели сам исходный программный текст" [9]. Изложенное связано с тем, что "идея доказательства правильности программ в значительной мере исчерпала себя, так как выяснилось, что в общем случае невозможно установить свойства результата работы программы или процедуры, не исполнив ее до конца... В теории вычислений доказывается, что в общем случае распознать определенные свойства в программе можно только динамически, прослеживая весь процесс вычисления при соответствующих входных воздействиях" [10].

Протоколирование можно выполнять с любой степенью подробности. При этом можно выделить две разновидности протоколов: "полные" и "короткие". В "полных" протоколах

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. "Короткие" протоколы содержат только события и инициируемые ими выходные воздействия, интересующие заказчика.

"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".

Каждый протокол можно рассматривать как сценарий работы системы при заданных входных воздействиях. Как указано выше, этот протокол строится автоматически во время исполнения программы, в то время как в известных технологиях сценарии (например, диаграммы последовательностей в UML [5]) предлагается строить вручную на этапе проектирования, что при сложном поведении программы практически невозможно.

Протоколы, получаемые на этапе отладки, используются в дальнейшем на этапе сертификации в роли контрольных примеров [8] для демонстрации того, что программа работает в соответствии с требованиями заказчика. Вопрос о количестве необходимых для сертификации протоколов остается открытым и требует дальнейших исследований.

3.1.5. Документирование

На этапе документирования для точного оформления результатов проектирования и разработки программы предлагается создавать и сдавать в архив документацию (по крайней мере в электронном виде), которая в общем случае может содержать следующие документы: 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.

После этого, при появлении любых изменений, возникающих в ходе дальнейших этапов жизненного цикла программы, весь комплект документации (по завершении каждого этапа) должен корректироваться.

Отметим, что 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.

4. Практическое применение предлагаемой технологии

4.1. Сравнение традиционного подхода с предлагаемым

Сравнение традиционного подхода с предлагаемым выполним на примере создания программного модуля управления элементом пользовательского интерфейса "тулбар" (рис. 13), реализующего следующие функции:

– перемещение тулбара при нажатой правой кнопке мыши;

– вывод меню тулбара нажатием и отпусканием правой кнопки мыши.

Рис. 13. Тулбар

Сначала построим этот модуль, используя традиционный событийный подход. При этом для выполнения заданных функций необходимо реализовать обработку следующих событий:

– нажатие правой кнопки мыши;

– отпускание правой кнопки мыши;

– перемещение мыши с нажатой правой кнопкой;

– выход курсора мыши за границу тулбара.

Ниже приведен текст модуля, состоящего из обработчиков перечисленных событий, который создан для работы под ОС QNX 4.25 и графической оболочкой Photon 1.14.

```
// Модуль, реализующий управление тулбаром.
// Традиционный подход.

//======
// Обработчик события нажатия кнопки мыши на тулбаре.
int toolbar_btn_press(PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo)
{
  toolbar_t               *tb = tb_data(widget) ;    // Указатель на данные тулбара.
  PhRect_t                *rect ;                    // Координаты курсора мыши.
  PhEvent_t               *event = cbinfo->event ;   // Произошедшее событие.
  PhPointerEvent_t  *edata = PhGetData( event ) ;    // Дополнительная информация о
                                                     // событии.

  if( edata->buttons&Ph_BUTTON_MENU )
  {
    // Была нажата правая кнопка мыши.

    rect = PhGetRects( event ) ;
    tb->drag_pos = rect->ul ;     // Запомнить координаты курсора мыши.
    PtWindowToFront( tb->wgt ) ;  // Переместить окно тулбара выше всех
                                  // остальных.
    tb->menu = 1 ;                // Запомнить, что была нажата правая кнопка мыши.
  } ;

  return( Pt_CONTINUE ) ;
} ;
```

```
//======
// Обработчик события отпускания кнопки мыши на тулбаре.
int toolbar_btn_release(PtWidget_t *widget,ApInfo_t *apinfo,PtCallbackInfo_t *cbinfo)
{
  toolbar_t               *tb = tb_data(widget) ;    // Указатель на данные тулбара.
  PhEvent_t               *event = cbinfo->event ;   // Произошедшее событие.
  PhPointerEvent_t  *edata = PhGetData( event ) ;    // Дополнительная информация о
                                                     // событии.

  if(    event->subtype==Ph_EV_RELEASE_REAL
      && edata->buttons&Ph_BUTTON_MENU )
  // Была отпущена правая кнопка мыши.
    if( tb->menu == 1 )
    // Перед этим была нажата правая кнопка мыши.
      ApCreateModule( ABM_toolbar_menu, NULL, NULL ) ;      // Отобразить меню.

  tb->menu = 0 ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Обработчик события перемещения мыши с нажатой кнопкой.
int toolbar_btn_move(PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo)
{
  toolbar_t               *tb = tb_data(widget) ;    // Указатель на данные тулбара.
  PhRect_t                *rect ;                    // Координаты события.
  PhEvent_t               *event = cbinfo->event ;   // Произошедшее событие.
  PhPointerEvent_t  *edata = PhGetData( event ) ;    // Дополнительная информация о
                                                     // событии.

  // Если не нажата правая кнопка мыши — ничего не делать.
  if( !edata->buttons&Ph_BUTTON_MENU ) return Pt_CONTINUE ;

  // Переместить тулбар вслед за курсором мыши.
  rect = PhGetRects( event ) ;
  tb->menu = 0 ;
  toolbar_move( tb, rect->ul.x - tb->drag_pos.x,
                    rect->ul.y - tb->drag_pos.y ) ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Обработчик события пересечения курсором мыши границы тулбара.
int toolbar_boundary(PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo)
{
  toolbar_t               *tb = tb_data(widget) ;    // Указатель на данные тулбара.

  tb->menu = 0 ;

  return( Pt_CONTINUE ) ;
} ;
```

На первый взгляд, кажется (как это обычно утверждается в литературе), что эти обработчики независимы. Однако это не так уже потому, что они предназначены для управления одним и тем же объектом (тулбаром), и имеет место их взаимосвязь за счет наличия общей управляющей переменной (menu). Из текста программы следует, что логика ее работы рассредоточена по обработчикам событий, что делает поведение модуля априори непредсказуемым.

Можно отметить и другой недостаток традиционного подхода. Для обеспечения понятности программы ее стараются делать самодокументирующейся. При этом вводятся комментарии на языке разработчика (в России обычно на русском) или заказчика. Также используются смысловые идентификаторы, которые по этой причине достаточно длинны и должны быть записаны на английском языке. Поэтому, во-первых, разработчик вынужден использовать английский язык в именах переменных и функций, что весьма неудобно, так как требует постоянного переключения мыслей с одного языка на другой. Во-вторых, при необходимости графического отображения результатов проектирования, что в последнее время все чаще требуется заказчиками, оно становится чрезвычайно громоздким, и поэтому либо все-таки не используется, либо не является досконально точным.

Этот же недостаток имеет место и в тех случаях, когда самодокументирующимся делают визуальный формализм. Примером этого являются диаграммы состояний (Statecharts), используемые в UML [5], в которых запись сложных логических выражений и большого числа выходных воздействий и их особенностей становится практически необозримой. Такая же ситуация имеет место и при использовании SDL-диаграмм, широко применяемых при программной реализации протоколов в телефонии [11].

Отметим также, что при традиционном написании текстов программ реализация функций входных и выходных воздействий обычно выполняется совместно с логикой, затрудняя ее понимание.

Перейдем к проектированию рассматриваемого модуля с применением описываемой технологии автоматного программирования.

Для каждого модуля, вместо самодокументирующейся программы, разрабатываются четыре документа, которые в совокупности, как будет показано ниже, решают вопрос о понятности поведения программы: словесное описание поведения модуля (например, перечень выполняемых модулем функций); схема связей автомата с его окружением (интерфейс автомата); граф переходов, однозначно и математически строго определяющий поведение автомата; текст программного модуля.

Используя словесное описание поведения модуля, формализуем перечень его входных и выходных воздействий в виде схемы связей автомата (рис. 14), указывая на ней для каждого из воздействий его источник (приемник), полное название (на языке разработчика) и идентификатор в виде буквы латинского алфавита с номером.

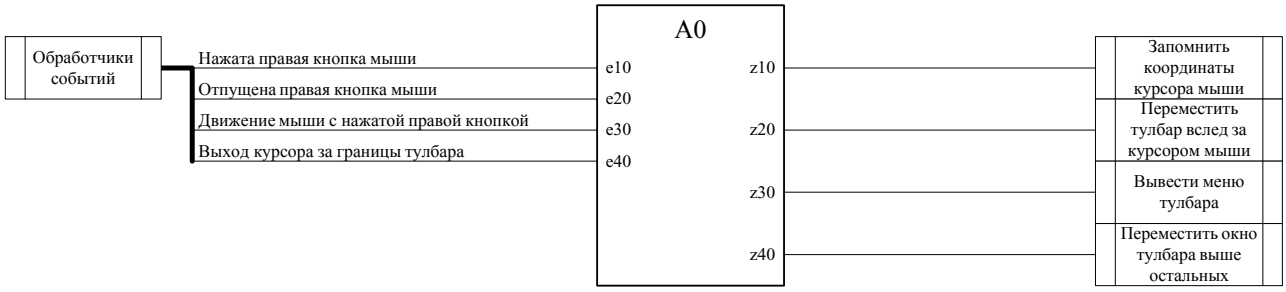

Рис. 14. Схема связей автомата управления тулбаром

Для построения графа переходов определим возможные состояния тулбара, которым сопоставим соответствующие состояния управляющего им автомата. Эти состояния являются "естественными", так как они связаны с физикой процесса управления: ожидание (начальное состояние) и перемещение.

Далее определим последовательности событий, инициирующие переходы между этими состояниями. После этого введем дополнительные состояния, "разделяющие" события в каждой из последовательностей. Дополнительное

состояние, возникающее на пути от состояния "ожидание" к состоянию "перемещение", назовем "готовность". Пронумеровав в графе переходов состояния (начиная с нуля), определим остальные переходы между состояниями и необходимые петли. Используя схему связей автомата, на графе переходов запишем условия переходов и действия, выполняемые в вершинах, на дугах и петлях. Построенный таким образом граф переходов смешанного автомата приведен на рис. 15.

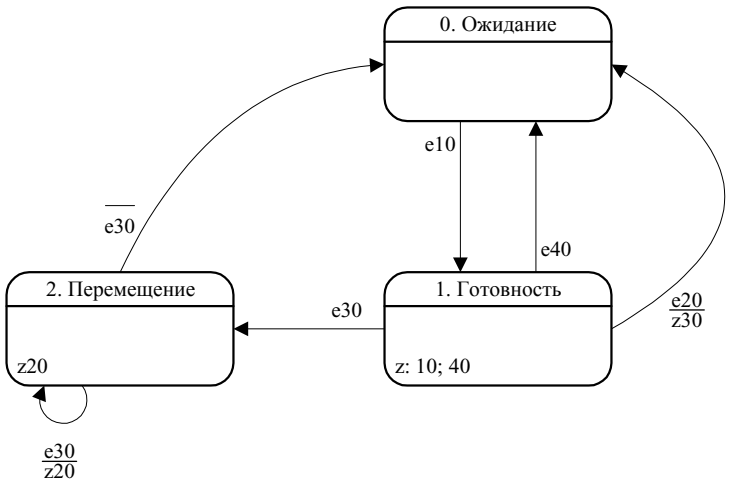

Рис. 15. Граф переходов автомата управления тулбаром

Наличие схемы связей позволяет понять смысл используемых в графе переходов идентификаторов, несмотря на их очень короткие и весьма абстрактные обозначения. Совместное изучение схемы связей и графа переходов позволяет даже не участвовавшему в разработке специалисту понять поведение создаваемого программного модуля.

Обратим внимание, что в построенном графе в вершине 2 выходное воздействие z20 формируется как собственно в вершине, так и на петле. Это означает, что указанное выходное воздействие формируется как при переходе в состояние 2, так и при поступлении события e30 при

нахождении автомата в этом состоянии, что поддерживается соответствующей реализацией.

Отметим, что при использовании SWITCH-технологии вместо термина **"логика программы"** (предполагающего работу с флагами) предлагается применять термин **"поведение программы"**, подразумевающий работу с состояниями.

Далее граф переходов (рис. 15) формально и изоморфно реализуется по шаблону в виде функции. Применяемый шаблон позволяет обеспечить внешнюю похожесть текста программного модуля на граф переходов. Построенная функция не требует пояснений (комментариев) к своему поведению, так как при использовании предлагаемой технологии текст программы не является основным (и тем более единственным) программным документом, а поведение однозначно и математически строго задается графом переходов. Эта функция не содержит реализации входных и выходных воздействий, а включает только их вызовы.

Получаемая при этом часть программы называется системонезависимой. Перейдем к построению ее системозависимой части, в соответствии с предлагаемой структурой событийных программ (рис. 7).

В программу добавляются обработчики событий, каждый из которых также реализован в виде функции и содержит вызов построенного автомата с передачей номера соответствующего события.

После этого, добавив в виде "заглушек" функции входных 'x' (если они имеются) и выходных 'z' воздействий, содержащие только вызовы функций протоколирования, можно уже на ранней стадии программной реализации получить действующий макет разрабатываемого модуля, что соответствует принципу пошаговой нисходящей разработки [8].

Программная реализация завершается разработкой располагаемых отдельно функций входных и выходных воздействий и используемых ими вспомогательных модулей. Эти функции обычно могут быть отлажены независимо.

```
// Модуль, реализующий управление тулбарами.
// Автоматный подход.

//======
// Обработчик события нажатия кнопки мыши на тулбаре.
int toolbar_btn_press( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  toolbar_t         *tb = tb_data(widget) ;          // Указатель на данные тулбара.
  PhEvent_t         *event = cbinfo->event ;         // Произошедшее событие.
  PhPointerEvent_t  *edata = PhGetData( event ) ;    // Дополнительная информация о
                                                     // событии.

  if( edata->buttons == Ph_BUTTON_MENU )
  {
    // Нажатая кнопка мыши — правая.
    // Вызвать управляющий автомат.
    A0( 10, tb, event ) ;
  } ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Обработчик события отпускания кнопки мыши на тулбаре.
int toolbar_btn_release( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  toolbar_t         *tb = tb_data(widget) ;          // Указатель на данные тулбара.
  PhEvent_t         *event = cbinfo->event ;         // Произошедшее событие.
  PhPointerEvent_t  *edata = PhGetData( event ) ;    // Дополнительная информация о
                                                     // событии.

  if( event->subtype == Ph_EV_RELEASE_REAL && edata->buttons == Ph_BUTTON_MENU )
  {
    // Отпущенная кнопка мыши — правая.
    // Вызвать управляющий автомат.
    A0( 20, tb, event ) ;
  } ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Обработчик события перемещения мыши с нажатой кнопкой.
int toolbar_btn_move( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  toolbar_t         *tb = tb_data(widget) ;          // Указатель на данные тулбара.
  PhEvent_t         *event = cbinfo->event ;         // Произошедшее событие.
  PhPointerEvent_t  *edata = PhGetData( event ) ;    // Дополнительная информация о
                                                     // событии.

  if( edata->buttons == Ph_BUTTON_MENU )
  {
    // Правая кнопка нажата.
    // Вызвать управляющий автомат.
    A0( 30, tb, event ) ;
  } ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Обработчик события пересечения курсором мыши границы тулбара.
int toolbar_boundary( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  toolbar_t         *tb = tb_data(widget) ;    // Указатель на данные тулбара.
```

```
  PhEvent_t         *event = cbinfo->event ;   // Произошедшее событие.

  if( event->subtype == Ph_EV_PTR_LEAVE )
  {
    // Курсор мыши вышел за границу тулбара.
    // Вызвать управляющий автомат.
    A0( 40, tb, event ) ;
  } ;

  return( Pt_CONTINUE ) ;
} ;

//======
// Функция, реализующая автомат управления тулбаром.
void A0( int e, toolbar_t *tb, PhEvent_t *event )
{
  int y_old = tb->y0 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( "A0", y_old, e ) ;
  #endif

  switch( tb->y0 )
  {
    case 0:
      if( e == 10 )                     tb->y0 = 1 ;
    break ;

    case 1:
      if( e == 20 ) { z30(tb) ;         tb->y0 = 0 ; }
      else
      if( e == 30 )                     tb->y0 = 2 ;
      else
      if( e == 40 )                     tb->y0 = 0 ;
    break ;

    case 2:
      if( e == 30 ) { z20(tb, event) ; }
      else
      if( e != 30 )                     tb->y0 = 0 ;
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ОШИБКА В A0: неизвестное состояние!", 0 ) ;
      #endif
    break ;
  } ;

  // Если состояние не изменилось - завершить выполнение функции.
  if( y_old == tb->y0 ) goto A0_end ;

  #ifdef GRAPH_TRANS_LOGGING
    log_trans( "A0", y_old, tb->y0 ) ;
  #endif

  switch( tb->y0 )
  {
    case 1:
      z10(tb, event) ; z40(tb) ;
    break ;

    case 2:
      z20(tb, event) ;
    break;
  } ;

  A0_end: ;
  #ifdef GRAPH_ENDS_LOGGING
    log_end( "A0", tb->y0, e ) ;
  #endif
} ;

//======
// Запомнить координаты курсора мыши.
void z10( toolbar_t *tb, PhEvent_t *event )
{
```

```
  PhRect_t      *rect = NULL ;    // Координаты курсора мыши.

  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z10. Запомнить координаты курсора мыши.", 0 ) ;
  #endif

  rect = PhGetRects( event ) ;
  tb->drag_pos = rect->ul ;
} ;

//======
// Переместить тулбар.
void z20( toolbar_t *tb, PhEvent_t *event )
{
  PhRect_t      *rect = NULL ;    // Координаты курсора мыши.

  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z20. Переместить тулбар.", 0 ) ;
  #endif

  rect = PhGetRects( event ) ;
  toolbar_move( tb, rect->ul.x - tb->drag_pos.x, rect->ul.y - tb->drag_pos.y ) ;
} ;

//======
// Вызвать меню тулбара.
void z30( toolbar_t *tb )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z30. Вызвать меню тулбара.", 0 ) ;
  #endif

  ApCreateModule( ABM_toolbar_menu, NULL, NULL ) ;
} ;

//======
// Переместить окно тулбара выше остальных.
void z40( toolbar_t *tb )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z40. Переместить окно тулбара выше остальных.", 0 ) ;
  #endif

  PtWindowToFront( tb->wgt ) ;
} ;
```

С силу того, что построенная на основе предложенного подхода программа имеет регулярную структуру, то вводя вызовы функций протоколирования в функции автоматов, входных и выходных воздействий имеется возможность автоматического получения истории выполнения программы в терминах автоматов в форме протокола.

"Полный" протокол с проверкой всех переходов автомата, реализующего алгоритм управления тулбаром

```
Обработка события "нажатие правой кнопки мыши"

16:44:58.543{ A0: в состоянии 0 запущен с событием e10
16:44:58.543T A0: перешел из состояния 0 в состояние 1
16:44:58.543* z10. Запомнить координаты курсора мыши.
16:44:58.543* z40. Переместить окно тулбара выше остальных.
16:44:58.543} A0: завершил обработку события e10 в состоянии 1

Обработка события "отпускание правой кнопки мыши" - вызов
меню тулбара

16:44:59.903{ A0: в состоянии 1 запущен с событием e20
```

```
16:44:59.903* z30. Вызвать меню тулбара.
16:44:59.903T A0: перешел из состояния 1 в состояние 0
16:44:59.903} A0: завершил обработку события e20 в состоянии 0

Обработка события "выход курсора мыши за границу тулбара"

16:44:59.903{ A0: в состоянии 0 запущен с событием e40
16:44:59.903} A0: завершил обработку события e40 в состоянии 0

Обработка события "нажатие правой кнопки мыши"

16:45:03.963{ A0: в состоянии 0 запущен с событием e10
16:45:03.963T A0: перешел из состояния 0 в состояние 1
16:45:03.963* z10. Запомнить координаты курсора мыши.
16:45:03.963* z40. Переместить окно тулбара выше остальных.
16:45:03.963} A0: завершил обработку события e10 в состоянии 1

Обработка события "выход курсора мыши за границу тулбара"

16:45:05.933{ A0: в состоянии 1 запущен с событием e40
16:45:05.933T A0: перешел из состояния 1 в состояние 0
16:45:05.933} A0: завершил обработку события e40 в состоянии 0

Обработка события "нажатие правой кнопки мыши"

16:45:10.482{ A0: в состоянии 0 запущен с событием e10
16:45:10.482T A0: перешел из состояния 0 в состояние 1
16:45:10.482* z10. Запомнить координаты курсора мыши.
16:45:10.482* z40. Переместить окно тулбара выше остальных.
16:45:10.482} A0: завершил обработку события e10 в состоянии 1

Обработка событий "перемещение мыши с нажатой кнопкой"

16:45:12.812{ A0: в состоянии 1 запущен с событием e30
16:45:12.812T A0: перешел из состояния 1 в состояние 2
16:45:12.812* z20. Переместить тулбар.
16:45:12.812} A0: завершил обработку события e30 в состоянии 2
16:45:12.852{ A0: в состоянии 2 запущен с событием e30
16:45:12.852* z20. Переместить тулбар.
16:45:12.852} A0: завершил обработку события e30 в состоянии 2

Обработка события "отпускание правой кнопки мыши"

16:45:15.812{ A0: в состоянии 2 запущен с событием e20
16:45:15.812T A0: перешел из состояния 2 в состояние 0
16:45:15.812} A0: завершил обработку события e20 в состоянии 0

Обработка события "выход курсора мыши за границу тулбара"

16:45:16.742{ A0: в состоянии 0 запущен с событием e40
16:45:16.742} A0: завершил обработку события e40 в состоянии 0

Обработка события "нажатие правой кнопки мыши"

16:45:18.992{ A0: в состоянии 0 запущен с событием e10
16:45:18.992T A0: перешел из состояния 0 в состояние 1
16:45:18.992* z10. Запомнить координаты курсора мыши.
16:45:18.992* z40. Переместить окно тулбара выше остальных.
16:45:18.992} A0: завершил обработку события e10 в состоянии 1

Обработка событий "перемещение мыши с нажатой кнопкой"

16:45:20.472{ A0: в состоянии 1 запущен с событием e30
16:45:20.472T A0: перешел из состояния 1 в состояние 2
16:45:20.472* z20. Переместить тулбар.
16:45:20.472} A0: завершил обработку события e30 в состоянии 2
16:45:21.192{ A0: в состоянии 2 запущен с событием e30
16:45:21.192* z20. Переместить тулбар.
16:45:21.192} A0: завершил обработку события e30 в состоянии 2

Обработка события "выход курсора мыши за границу тулбара" -
прекращение перемещения

16:45:21.232{ A0: в состоянии 2 запущен с событием e40
16:45:21.232T A0: перешел из состояния 2 в состояние 0
16:45:21.232} A0: завершил обработку события e40 в состоянии 0
```

Из изложенного в настоящем разделе следует, что предлагаемый подход резко повышает "понимаемость" разрабатываемой программы. При его использовании, в отличие от традиционных подходов, "сходятся концы с концами" — автоматы применяются для спецификации, программирования, протоколирования и документирования.

4.2. Подсистема управления печатью

Рассмотрим применение предлагаемого подхода на более сложном примере, состоящем в разработке подсистемы управления печатью.

Подсистема функционирует под управлением ОС QNX версии 4.25 и предназначена для реализации различных режимов печати истории функционирования системы с возможностью их предварительного просмотра в среде графической оболочки Photon 1.14.

Эта подсистема состоит из двух частей:

– менеджера печати, функционирующего как отдельный процесс (автомат выполнения печати);

– модуля контроля режима печати, внедряемого в использующее его программное обеспечение в виде библиотеки (автомат контроля режима печати).

Модуль контроля режима печати реализует следующие функции:

– печать строк с информацией о происходящих в системе событиях в режиме автоматического функционирования системы;

– предварительный просмотр отдельных фрагментов истории работы системы и их печать по требованию оператора;

– печать выбранных фрагментов истории работы системы по требованию оператора или при срабатывании таймера.

Менеджер печати и модуль контроля режима печати взаимодействуют асинхронно через два файла, расположенные на RAM-диске (рис. 16).

Некоторые функции системозависимой части модуля контроля печати также реализован с помощью автомата (автомат буферизации предварительного просмотра).

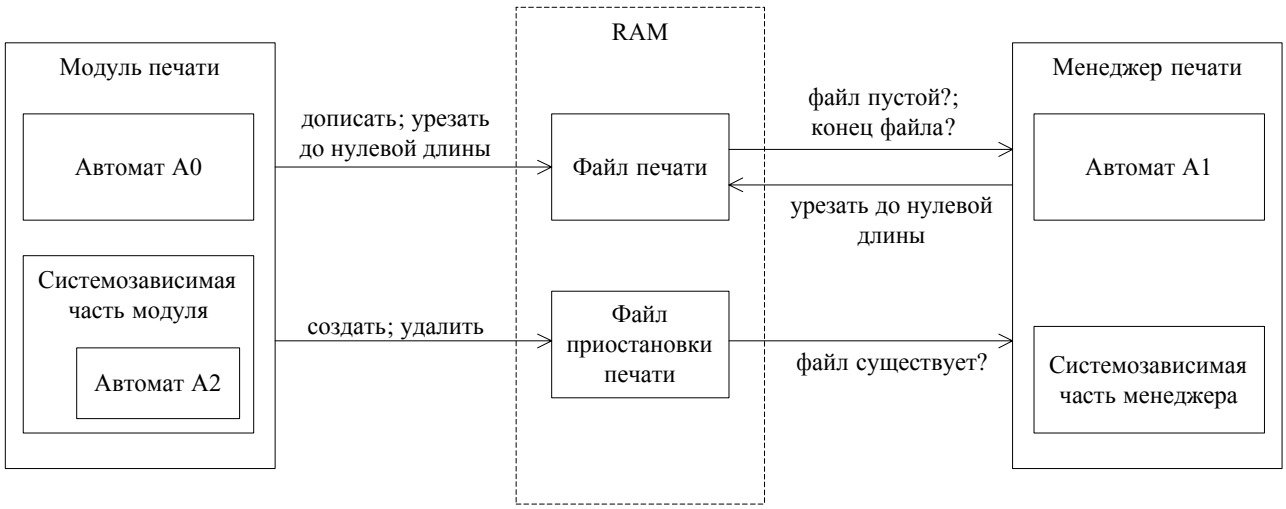

Рис. 16. Структурная схема программного обеспечения подсистемы управления печатью

Отметим, что рассмотренный пример является достаточно "простым" в рамках предложенной технологии, так как в нем автоматы не взаимодействуют друг с другом ни по одному из трех указанных выше способов (вложенность, вызываемость, обмен номерами состояний).

Рассмотрим более подробно выделенные автоматы, приведя для каждого из них четыре документа: словесное описание; схему связей; граф переходов; текст функции, реализующей автомат.

4.2.1. Автомат контроля режима печати

4.2.1.1. *Словесное описание*

Автомат контроля режима печати (A0) предназначен для обеспечения предварительного просмотра и печати файлов в различных режимах работы системы.

При переводе системы в автоматический режим работы (событие e12) автомат переходит в состояние "Автоматический". При этом в очередь печати добавляется заголовок таблицы происходящих в системе событий. Добавление в очередь строки с информацией о произошедшем в системе событии осуществляется при вызове интерфейсной функции print_on_event(), в качестве параметра которой передается указанная строка. Эта функция запускает автомат с событием, означающим необходимость напечатать очередную строку (событие e110). Если размер файла печати не превышает заданного (переменная x70), переданная строка добавляется в этот файл, являющийся очередью печати и обрабатываемый менеджером печати, а функция print_on_event() возвращает 0. В противном случае функция print_on_event() возвращает 2.

При переводе системы в ручной режим работы (событие e13), автомат переходит в состояние "Ручной общий", в котором при нажатии общей кнопки "ПЕЧАТЬ" (событие e30) или срабатывании таймера печати по времени (событие e80) фрагменты истории работы системы в соответствии с текущим выбором, выполненным оператором на соответствующем видеокадре, добавляются в очередь печати.

При нажатии оператором на указанном видеокадре одной из кнопок, соответствующих различным фрагментам, автомат вызывается с событием e40. При этом в качестве параметра функция автомата получает указатель на структуру данных с информацией о диалоге, в котором должен осуществляться

предварительный просмотр выбранного фрагмента. После этого, при условии успешного открытия файла данного фрагмента (переменная x10), автомат переходит в состояние "Ручной частный", в котором открывается указанный диалог и инициируется предварительный просмотр. При нажатии оператором в диалоге предварительного просмотра кнопки "ПЕЧАТЬ" (событие e30), выбранный фрагмент добавляется в очередь печати.

Возврат из состояния "Ручной частный" в состояние "Ручной общий" осуществляется при закрытии диалога предварительного просмотра (событие e11).

Схема связей автомата и граф переходов приведены на рис. 17, 18.

4.2.1.2. *Схема связей*

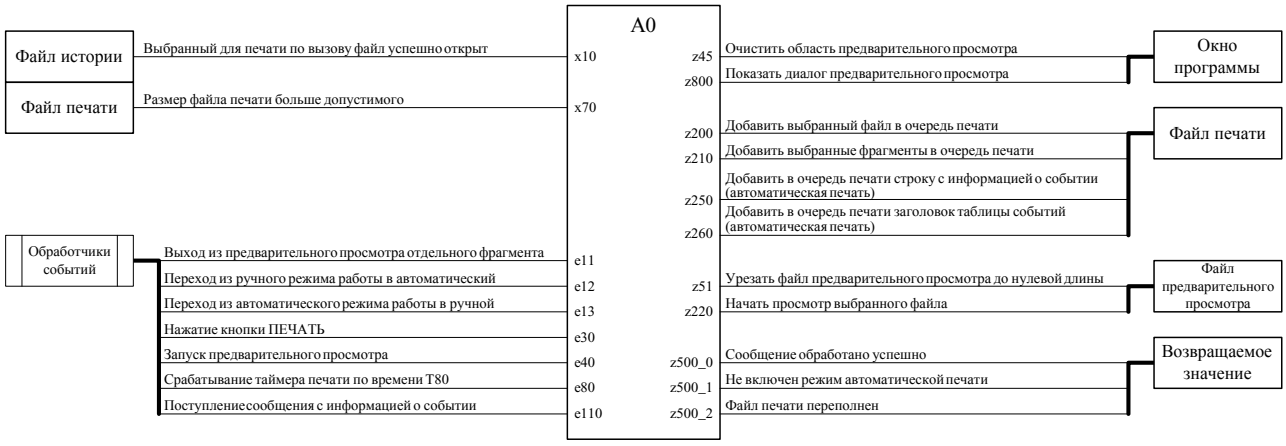

Рис. 17. Схема связей автомата контроля режима печати A0

4.2.1.3. *Граф переходов*

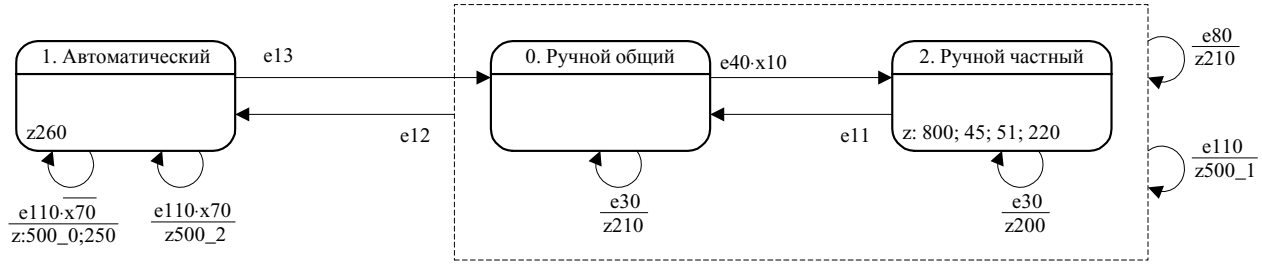

Рис. 18. Граф переходов автомата контроля режима печати A0

4.2.1.4. *Текст функции*

Текст функции, реализующей автомат A0, приведен ниже. Отметим, что в состоянии 1 для устранения описанной ранее проблемы "риска" опрос входной переменной x70 выполняется однократно, а ее значение запоминается во вспомогательной переменной X70.

```
#include <sys/types.h>
#include "photon_stuff.h"
#include "log.h"
#include "defines.h"

static int y0 = 0 ;

void A0( int e, preview_dialog_t *d )
{
  int y_old = y0 ;

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

  switch( y0 )
  {
    case 0:
      if( e == 80 )  { z210() ; }
      else
      if( e == 110 ) { z500_1() ; }
      else
      if( e == 12 )                  y0 = 1 ;
      else
      if( e == 30 )  { z210() ; }
      else
      if( e == 40 && x10(d))         y0 = 2 ;
    break ;

    case 1:
    {
      int    X70 = x70() ;

      if( e == 13 )                  y0 = 0 ;
      else
      if( e == 110 && X70 ) { z500_2() ; }
      else
      if( e == 110 && !X70 ) { z500_0() ; z250() ; }
    }
    break ;
```

```
    case 2:
    {
      if( e == 80 )  { z210() ; }
      else
      if( e == 110 ) { z500_1() ; }
      else
      if( e == 12 )                  y0 = 1 ;
      else
      if( e == 11 )                  y0 = 0 ;
      else
      if( e == 30 )  { z200(d) ; }
    }
    break ;

    default:
      #ifdef A0_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ERROR IN A0: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == y0 ) goto A0_end ;

  {
    #ifdef A0_TRANS_LOGGING
      log_trans( "A0", y_old, y0 ) ;
    #endif
  } ;

  switch( y0 )
  {
    case 1:
      z260() ;
    break ;

    case 2:
      z800(d) ; z45(d) ; z51() ; z220(d) ;
    break ;
  } ;

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

4.2.2. Автомат буферизации предварительного просмотра

4.2.2.1. *Словесное описание*

Автомат буферизации предварительного просмотра (A2) предназначен для ускорения графического вывода при выполнении предварительного просмотра фрагментов истории работы системы. Автомат вызывается из функции перерисовки окна предварительного просмотра.

Необходимость в разработке нестандартных средств просмотра файла объясняется тем, что в используемом для предварительного просмотра графическом окне невозможно при помощи функций отображения текста отобразить символы псевдографики, которые могут встречаться в

просматриваемых файлах. Поэтому используемые символы псевдографики отображаются при помощи функций рисования линий.

Для ускорения перерисовки окна предварительного просмотра содержимое просматриваемого файла выводится, по возможности, блоками. При этом:

– непрерывные последовательности текстовых символов, не содержащие пробелов, выводятся одним блоком. Корректно выводить последовательности текстовых символов, содержащие пробелы, невозможно из-за ошибки в функции рисования текста, проявляющейся в том, что ширина символа "пробел" не совпадает с шириной обычного текстового символа;

– непрерывные последовательности пробелов не выводятся;

– непрерывные последовательности псевдографических символов "горизонтальная линия" выводятся одним вызовом функции рисования линии;

– другие символы псевдографики выводятся посимвольно.

Автомат выполняет функцию распознавателя указанных последовательностей.

Состояние "Вывод символа" соответствует посимвольному выводу.

Состояние "Текст" соответствует накоплению непрерывной последовательности текстовых символов.

Состояние "Линия" соответствует накоплению непрерывной последовательности псевдографических символов "горизонтальная линия".

Для обработки очередного символа автомат вызывается с событием e0, а при завершении обработки видимой части файла автомат вызывается с событием e500, по которому происходит вывод накопленного в буфере содержимого. Кроме

номера события автомату также передается код обрабатываемого символа и номера строки и столбца, соответствующие его местоположению.

Для ускорения работы некоторые функции входных переменных и выходных воздействий реализованы внутри функции автомата.

Схема связей и граф перехода автомата приведены на рис. 19, 20.

4.2.2.2. *Схема связей*

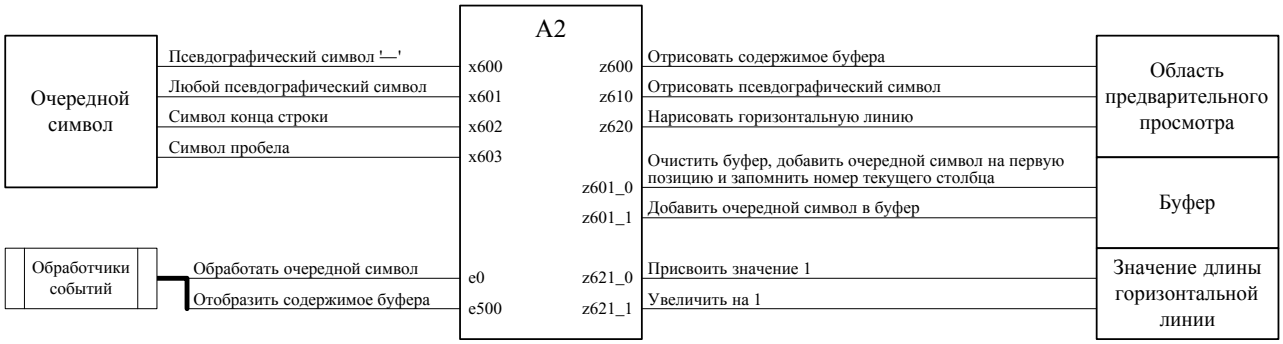

Рис. 19. Схема связей автомата буферизации предварительного просмотра A2

4.2.2.3. *Граф переходов*

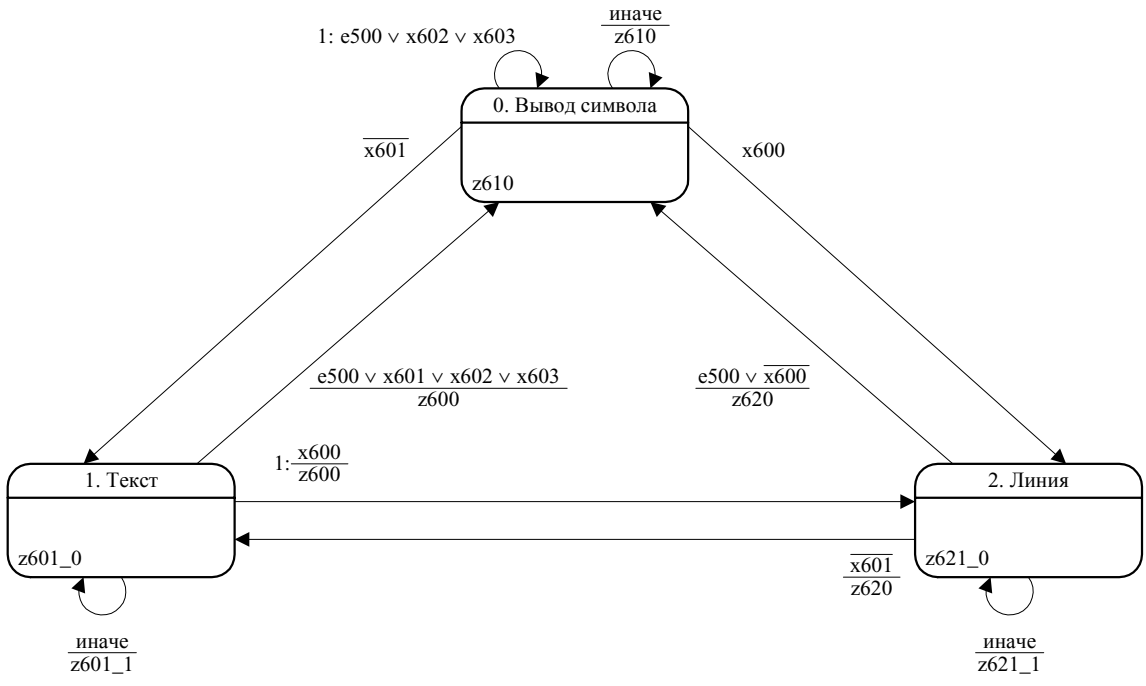

Рис. 20. Граф переходов автомата буферизации предварительного просмотра A2

4.2.2.4. *Текст функции*

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

void A2( int e, char c, int line_num, int col_num )
{
  static int    y2 = 0 ;
  int           y_old = y2 ;

  static char   buf[line_width+10] = "" ;
  static int    buf_len = 0 ;
  static int    line_len = 0 ;
  static int    start_col = 0 ;

  #ifdef A2_SHOW_SYMBOL
  {
    char    str[100] = "" ;

    sprintf( str, "Получен символ %d (`%c`)", c, c ) ;
    log_write( '!', str, 0 ) ;
  }
  #endif

  #ifdef A2_BEGIN_LOGGING
    log_begin( "A2", y_old, e ) ;
  #endif

  switch( y2 )
  {
    case 0:
      if( e == 500 || (c == 0x0A || c == 0x0D) || (c == ' ')) ;
      else
      if( c < 176 || c > 223 )                                  y2 = 1 ;
      else
      if( c == '-' )                                            y2 = 2 ;
      else
      { z610( c, line_num, col_num ) ; }
    break ;

    case 1:
      if( c == '-' )
      { z600( line_num, col_num, buf, buf_len, start_col ) ;    y2 = 2 ; }
      else
      if( e == 500 || (c >= 176 && c <= 223) || (c == 0x0A || c == 0x0D) || (c == ' ') )
        { z600( line_num, col_num, buf, buf_len, start_col ) ;  y2 = 0 ; }
      else
      { buf[buf_len++] = c ; buf[buf_len] = 0 ; }

    break ;

    case 2:
      if( e == 500 || c != '-' )
      { z620( line_num, col_num, line_len, start_col ) ;        y2 = 0 ; }
      else
      if( c < 176 || c > 223 )
      { z620( line_num, col_num, line_len, start_col ) ;        y2 = 1 ; }
      else
      { line_len++ ; }
    break ;

    default:
      #ifdef A2_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ERROR IN A2: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == y2 ) goto A2_end ;

    #ifdef A2_TRANS_LOGGING
      log_trans( "A2", y_old, y2 ) ;
    #endif
```

```
  switch( y2 )
  {
    case 0:
      z610( c, line_num, col_num ) ;
    break ;

    case 1:
      start_col = col_num ;
      buf[0] = c ; buf_len = 1 ; buf[1] = 0 ;
    break ;

    case 2:
      start_col = col_num ;
      line_len = 1 ;
    break ;
  } ;

  A2_end: ;
  #ifdef A2_END_LOGGING
    log_end( "A2", y2, e ) ;
  #endif
}
```

4.2.3. Автомат вывода на печать

4.2.3.1. *Словесное описание*

Автомат вывода на печать описывает поведение менеджера печати, функционирующего как отдельный процесс и предназначенного для непосредственной передачи на принтер распечатываемых документов.

Запуск автомата происходит при срабатывании таймера (событие e70), период которого задается в программе. Этот таймер запускается в режиме генератора синхроимпульсов при инициализации программы.

Автомат следит за двумя файлами:

– файлом, наличие которого является сигналом приостановки печати (переменная x60);

– файлом печати, в который добавляются выдаваемые на печать документы (переменные x10, x50).

Так как эти файлы физически находятся в оперативной памяти (на ram-диске), периодическая проверка их состояния не должна заметно влиять на общую производительность системы, особенно учитывая то, что менеджер печати запускается с низким приоритетом, задаваемым в программе при инициализации.

При отсутствии команды на приостановку печати (отсутствует файл приостановки печати) поведение автомата может быть описано следующим образом: при добавлении нового документа в файл печати, автомат переходит в состояние "Печать", в котором при срабатывании таймера в порт принтера отправляется очередной фрагмент распечатываемого документа. Размер этого фрагмента задается в программе и выбирается в зависимости от заданного периода срабатывания таймера и быстродействия принтера. При достижении конца файла печати автомат переходит в состояние "Принтер свободен", при этом файл печати урезается до нулевой длины. Если во время печати файл печати был урезан до нулевой длины, это является командой прекращения печати, после чего автомат также переходит в состояние "Принтер свободен".

Для устранения возможности неправильной работы, связанной с одновременной модификацией файла печати менеджером печати и другими процессами, используется механизм блокировки файлов.

Схема связей и граф перехода автомата приведены на рис. 21, 22.

4.2.3.2. *Схема связей*

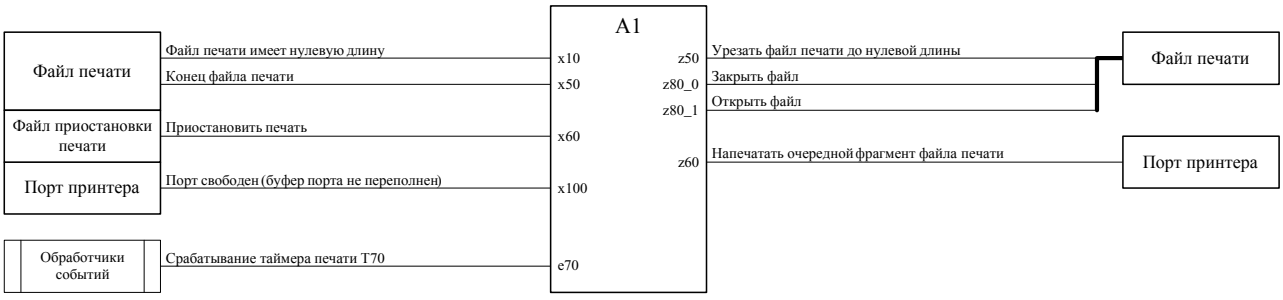

Рис. 21. Схема связей автомата вывода на печать A1

4.2.3.3. *Граф переходов*

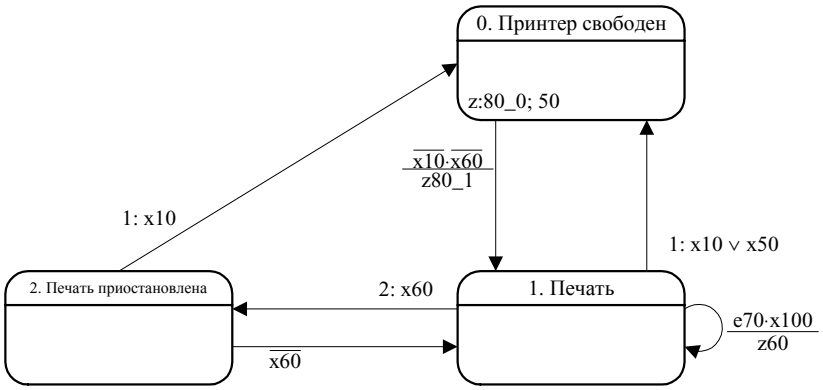

Рис. 22. Граф переходов автомата вывода на печать A1

4.2.3.4. *Текст функции*

```
#include <sys/types.h>
#include "log.h"
#include "defines.h"
#include "x.h"
#include "z.h"

static int y1 = 0 ;

void A1( int e )
{
  int y_old = y1 ;

  #ifdef A1_BEGIN_LOGGING
    log_begin( "A1", y_old, e ) ;
  #endif

  switch( y1 )
  {
    case 0:
      if( !x10() && !x60() )  { z80_1() ;            y1 = 1 ; } ;
    break ;

    case 1:
      if( x10() || x50() )                          y1 = 0 ;
      else
      if( x60() )                                   y1 = 2 ;
      else
      if( e == 70 && x100() ) { z60() ; }
    break ;

    case 2:
      if( x10() )                                   y1 = 0 ;
      else
      if( !x60() )                                  y1 = 1 ;
    break ;

    default:
      #ifdef A1_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, "ERROR IN A1: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == y1 ) goto A1_end ;
```

```
  {
    #ifdef A1_TRANS_LOGGING
      log_trans( "A1", y_old, y1 ) ;
    #endif
  } ;

  switch( y1 )
  {
    case 0:
      z80_0() ; z50() ;
    break ;
  } ;

  A1_end: ;
  #ifdef A1_END_LOGGING
    log_end( "A1", y1, e ) ;
  #endif
} ;
```

4.2.4. Системозависимая часть модуля печати

Системозависимая часть модуля печати состоит из файлов, содержащих:

– реализацию входных переменных;

– реализацию выходные воздействия;

– обработчики событий нажатия кнопок;

– обработчик события закрытия диалога предварительного просмотра печатаемых фрагментов;

– обработчик события срабатывания таймера;

– обработчик события печати строки в режиме автоматической печати;

– функцию инициализации;

– функции манипулирования файлами;

– функции, реализующие предварительный просмотр.

Приведем текст файла, содержащего реализацию входных переменных:

```
#include "photon_stuff.h"
#include "defines.h"
#include "log.h"
#include <sys/dev.h>

extern char     print_preview_file_name[] ;
extern char     print_spool_file_name[]  ;
extern char     print_current_dir[] ;
extern int      print_preview_file_limit ;
extern int      print_spool_file_limit ;

int x10( preview_dialog_t *d )
{
  int       result ;
```

```
  FILE      *print_preview_file = NULL ;

  if( d->file_name )
  {
    print_preview_file = fopen( d->file_name, "r" ) ;
    result = print_preview_file != NULL ;
    fclose( print_preview_file ) ;
  }
  else
    result = 0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x10 - выбранный для печати файл существует", result ) ;
  #endif

  return result ;
} ;

int x70()
{
  int           result = 1 ;

  result = print_file_size( print_spool_file_name ) > print_spool_file_limit ;

  #ifdef INPUTS_LOGGING
    log_input( "x70 - размер файла печати больше допустимого", result ) ;
  #endif

  return result ;
} ;

int x600( char c )
{
  int           result = 1 ;

  result = c == '-' ;

  #ifdef INPUTS_LOGGING
    log_input( "x600 - получен псевдографический символ горизонтальной линии", result ) ;
  #endif

  return result ;
} ;

int x601( char c )
{
  int           result = 1 ;

  result = c >= 176 && c <= 223 ;

  #ifdef INPUTS_LOGGING
    log_input( "x601 - получен псевдографический символ", result ) ;
  #endif

  return result ;
} ;

int x602( char c )
{
  int           result = 1 ;

  result = c == 0x0A || c == 0x0D ;

  #ifdef INPUTS_LOGGING
    log_input( "x602 - получен символ конца строки", result ) ;
  #endif

  return result ;
} ;

int x603( char c )
{
  int           result = 1 ;

  result = c == ' ' ;
```

```
  #ifdef INPUTS_LOGGING
    log_input( "x603 - получен символ конца пробел", result ) ;
  #endif

  return result ;
} ;
```

Часть текста файла, содержащего реализацию выходных воздействия, приведена ниже:

```
#include "photon_stuff.h"
#include "log.h"
#include "defines.h"
#include "nls_api.h"
#include <sys/kernel.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <share.h>

extern char         print_suspend_semaphor_file_name[] ;
extern char         print_spool_file_name[] ;
extern char         print_preview_file_name[] ;
extern int          print_buffer_size ;
extern int          print_on_event_result ;
extern PhRect_t     symbol_size ;
extern PhPoint_t    symbol_center ;

extern preview_dialog_t *active_dialog ;
extern preview_dialog_t dialogs[] ;

void z45( preview_dialog_t *d )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z45. Очистить окно предварительного просмотра.", 0 ) ;
  #endif

  print_preview_stop_preview( d ) ;
} ;

void z50()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z50. Урезать файл спулинга.", 0 ) ;
  #endif

  {
    int     filedes = -1 ;

    filedes = sopen( print_spool_file_name, O_WRONLY | O_CREAT | O_TRUNC,
                     SH_DENYWR, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP ) ;

    if( filedes != -1 )
      close( filedes ) ;
   }
} ;

void z51()
{
  FILE    *file ;

  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z51. Урезать файл предварительного просмотра.", 0 ) ;
  #endif

  file = fopen( print_preview_file_name, "w" ) ;
  if( file )
    fclose( file ) ;
} ;

void z55_0()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z55_0. Disable suspending.", 0 ) ;
  #endif
```

```
  remove( print_suspend_semaphor_file_name ) ;
} ;

void z55_1()
{
  struct stat   buf ;

  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z55_1. Enable or disable suspending.", 0 ) ;
  #endif

  if( stat( print_suspend_semaphor_file_name, &buf ) == 0 )
  {
    // Suspend file exists.
    remove( print_suspend_semaphor_file_name ) ;
  }
  else
  {
    FILE    *file ;

    file = fopen( print_suspend_semaphor_file_name, "w" ) ;
    if( file )
      fclose( file ) ;
  }
} ;

void z200( preview_dialog_t *d )
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z200. Добавить выбранный файл в очередь печати.", 0 ) ;
  #endif

  print_files_protocol_print( d->file_name ) ;
} ;

void z210()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z210. Добавить выбранные фрагменты в очередь печати.", 0 ) ;
  #endif

  print_files_protocols_print() ;
} ;
```

Приведем часть текста файла, содержащего обработчики событий нажатия кнопок:

```
#include "photon_stuff.h"

extern preview_dialog_t     dialogs[] ;
extern preview_dialog_t     *active_dialog ;

int print_buttons( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  if(PtWidgetParent(ABW_prot_anpar)!=NULL)
  {
   active_dialog = &dialogs[0] ;
    A0( 40, active_dialog ) ;
  }
  else


  if(PtWidgetParent(ABW_prot_ost)!=NULL)
  {
    active_dialog = &dialogs[1] ;
    A0( 40, active_dialog ) ;
  }
  else
  //...
  if(PtWidgetParent(ABW_kzip)!=NULL)
  {
    active_dialog = &dialogs[31];
    A0( 40, active_dialog ) ;
```

```
  }
  else
  if( widget == ABW_print_clear_spool )
  {
    z50() ;
  }
  else
  if( widget == ABW_print_suspend_printing)
  {
    z55_1() ;
  }

  if( widget == ABW_F6)
  {
    A0( 30, active_dialog ) ;
  }

  return( Pt_CONTINUE );
}
```

Приведем текст обработчика события закрытия диалога предварительного просмотра печатаемых фрагментов:

```
#include "photon_stuff.h"

extern preview_dialog_t     *active_dialog ;

int preview_close( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
  active_dialog = NULL ;
  A0( 11, NULL ) ;

  return( Pt_CONTINUE ) ;
} ;
```

Приведем текст обработчика события срабатывания таймера:

```
#include "photon_stuff.h"

int T80( void *data, pid_t pid, void *msg, size_t msg_size )
{
  A0( 80, NULL ) ;

  return Pt_CONTINUE ;
} ;
```

Приведем текст обработчика события печати строки в режиме автоматической печати:

```
// Возвращает:
//  0 - строка отправлена на печать;
//  1 - не включен автоматический режим печати;
//  2 - файл печати переполнен.
//
int print_on_event( const char * const str )
{
  print_event_string[0] = 0 ;
  strncpy( print_event_string, str, line_width ) ;
  print_event_string[line_width] = 0 ;
```

```
  A0(110, NULL ) ;

  return print_on_event_result ;
}
```

Приведем текст основной функции, реализующей предварительный просмотр:

```
//======
// Нарисовать предварительный просмотр.
//
int print_draw_preview( PtWidget_t *widget )
{
  int       line_num = 0, file_line_num = 0 ;
  PtArg_t   args[1] ;
  int       *scroll_pos ;

  if( !print_preview_is_shown) return 0 ;
  if( widget != active_dialog->preview_wgt ) return 0 ;

  // Получить положение скролбара.
  PtSetArg( args, Pt_ARG_SCROLL_POSITION, &scroll_pos, 0 ) ;
  PtGetResources( active_dialog->scrollbar_wgt, 1, args ) ;

  PgSetStrokeColor( preview_color ) ;
  PgSetTextColor( preview_color ) ;
  PgSetFont( preview_font ) ;

  //* Отобразить соответствующую часть файла предварительного просмотра.
  line_num = 0 ;
  file_line_num = -1 ;
  while( fgets( str, 999, print_preview_file ) )
  {
    int       col_num = 0 ;

    file_line_num++ ;

    // Вышли за пределы видимой области - закончить чтение файла.
    if( file_line_num > *scroll_pos + lines_on_screen ) break ;

    // Еще не дошли до видимой области - продолжать считывание.
    if( file_line_num < *scroll_pos ) continue ;

    str[line_width] = 0 ;

    while( str[col_num] != 0 )
    {
      //print_draw_symbol( str[col_num], line_num, col_num ) ;
      A2( 0, str[col_num], line_num, col_num ) ;

      col_num++ ;
    }

    A2( 500, 0, line_num, col_num ) ;

    line_num++ ;
  }

  fseek( print_preview_file, 0, SEEK_SET ) ;

  return 1 ;
}
```

4.2.5. Системозависимая часть менеджера печати

Системозависимая часть менеджера печати состоит из файлов, содержащих:

– реализацию входных переменных;

– реализацию выходных воздействий;

– функции инициализации программы;

– функции протоколирования.

Приведем текст файла, содержащего реализацию входных переменных:

```
#include <sys/dev.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include "defines.h"
#include "log.h"

extern char     print_suspend_semaphor_file_name[] ;
extern char     print_spool_file_name[]  ;
extern char     print_current_dir[] ;

extern FILE     *print_spool_file ;
extern int      print_port_desc ;

int x10()
{
  int           result = 0 ;
  struct stat   buf ;

  if( stat( print_spool_file_name, &buf ) == 0 )
    result = buf.st_size == 0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x10 - файл печати имеет нулевую длину", result ) ;
  #endif

  return result ;
} ;

int x50()
{
  int       result = 1 ;

  if( print_spool_file )
  {
    int         read_len = 0, read_buf ;
    long int    pos ;

    pos = ftell( print_spool_file ) ;
    read_len = fread( &read_buf, 1, 1, print_spool_file ) ;
    fseek( print_spool_file, pos, SEEK_SET ) ;

    result = read_len == 0 ;
  }

  #ifdef INPUTS_LOGGING
    log_input( "x50 - конец файла спулинга", result ) ;
  #endif

  return result ;
} ;

int x60()
{
  int           result = 0 ;
  struct stat   buf ;

  result = stat( print_suspend_semaphor_file_name, &buf ) == 0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x60 - suspend enabled", result ) ;
  #endif
```

```
  return result ;
} ;

int x100()
{
  int           result = 1 ;

  result = dev_state( print_port_desc, _DEV_EVENT_OUTPUT, _DEV_EVENT_OUTPUT ) ? 1:0 ;

  #ifdef INPUTS_LOGGING
    log_input( "x100 - порт принтера свободен", result ) ;
  #endif

  return result ;
} ;
```

Приведем текст файла, содержащего реализацию выходных воздействий:

```
#include <sys/kernel.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <share.h>
#include <unistd.h>
#include "log.h"
#include "defines.h"
#include "nls_api.h"

extern char     print_spool_file_name[]  ;
extern FILE     *print_spool_file ;
extern int      print_buffer_size ;
extern int      print_port_desc ;

void z50()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z50. Урезать файл спулинга.", 0 ) ;
  #endif

  {
    int     filedes = -1 ;

    filedes = sopen( print_spool_file_name, O_WRONLY | O_CREAT | O_TRUNC,
                     SH_DENYWR, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP ) ;

    if( filedes != -1 )
      close( filedes ) ;
   }
} ;

void z60()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z60. Напечатать очередной фрагмент файла спулинга.", 0 ) ;
  #endif

  if( print_spool_file )
  {
    char        buf[1000] = "" ;
    int         read_len = 0 ;
    long int    pos ;

    pos = ftell( print_spool_file ) ;
    read_len = fread( buf, 1, print_buffer_size, print_spool_file ) ;
    fseek( print_spool_file, pos+read_len, SEEK_SET ) ;

    write( print_port_desc, buf, read_len ) ;
    flushall() ;
  } ;
} ;
```

```
void z80_0()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z80_0. Закрыть файл спулинга.", 0 ) ;
  #endif

  if( print_spool_file )
  {
    fclose( print_spool_file ) ;
    print_spool_file = NULL ;
  }
} ;

void z80_1()
{
  #ifdef ACTIONS_LOGGING
    log_write( LOG_ACTION, "z80_1. Открыть файл спулинга.", 0 ) ;
  #endif

  print_spool_file = fopen( print_spool_file_name, "r" ) ;
} ;
```

Приведем текст файла, содержащего функцию инициализации программы:

```
#include <sys/sched.h>
#include <sys/stat.h>
#include <sys/dev.h>
#include <sys/proxy.h>
#include <sys/kernel.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include "z.h"
#include "log.h"
#include "A.h"

//*********************************************************************
//* Параметры для настройки менеджера печати.
//*

// Имя порта, к которому подключен принтер.
const char          print_port_name[PATH_MAX] = "/dev/ser2" ;
//const char          print_port_name[PATH_MAX] = "/hd/print.test" ;

// Имя временного файла для печати - файла спулинга (с полным путем).
const char          print_spool_file_name[PATH_MAX] = "/hd/print.spool" ;

const char          print_suspend_semaphor_file_name[PATH_MAX] = "/hd/print.suspend" ;

// Период, с которым будет осуществляться посылка символьных пакетов на принтер (сек).
const int           T70_interval = 1 ;

// Количество символов в пакете
// (значение должно соответствовать быстродействию принтера!).
const int           print_buffer_size = 50 ;

// Дескриптор временного файла для печати.
FILE                *print_spool_file = NULL ;

// Дескриптор порта, к которому подключен принтер.
int                 print_port_desc = -1 ;

void main()
{
  pid_t                   proxy_pid = -1, client_pid = -1 ;

  // Установить приоритет.
  setprio( 0, 1 ) ;
```

```
  // Подключится к порту.
  print_port_desc = open( print_port_name, O_RDWR ) ;
  if( print_port_desc == -1 )
  {
    perror( "Error opening printer port" ) ;
    exit( -1 ) ;
  }

  dev_state( print_port_desc, 0, _DEV_EVENT_OUTPUT ) ;
  dev_osize( print_port_desc, print_buffer_size ) ;

  // Запустить таймер печати.
  {
    timer_t                 timer_id = -1 ;
    struct sigevent         t_event ;
    struct itimerspec       t_value ;

    proxy_pid = qnx_proxy_attach( 0, 0, 0, -1 ) ;
    if( proxy_pid == -1 )
    {
      log_write( '!', "ERROR IN TIMERS!!!", errno ) ;
      exit(-1) ;
    } ;

    t_event.sigev_signo = -proxy_pid ;
    timer_id = timer_create( CLOCK_REALTIME, &t_event ) ;
    if( timer_id == -1 )
    {
      log_write( '!', "ERROR IN TIMERS!!!", errno ) ;
      exit(-1) ;
    } ;

    t_value.it_value.tv_sec = T70_interval ;
    t_value.it_value.tv_nsec = 0 ;
    t_value.it_interval.tv_sec = T70_interval ;
    t_value.it_interval.tv_nsec = 0 ;
    timer_settime( timer_id, 0, &t_value, NULL ) ;
  }

  // Произвести инициализацию.
  z50() ;

  // Основной цикл работы менеджера печати.
  for(;;)
  {
    int     msg ;

    client_pid = Receive( 0, &msg, sizeof( msg )) ;
    Reply( client_pid, NULL, 0 ) ;

    if( client_pid == proxy_pid )
    {
      A1( 70 ) ;
      continue ;
    }
  }
}
```

4.2.6. Протоколы функционирования модуля печати

В случае системы, для которой разрабатывался модуль печати, протоколы являлись единственным эффективным средством отладки, так как размер исполняемого файла системы, в который внедрялся модуль печати, составлял 5,5Мб, а используемый отладчик не мог загружать файлы такого большого объема.

Благодаря использованию протоколов удалось устранить ошибки во взаимодействии системы с модулем печати. Так, например, из приведенного ниже протокола следует (строки со временем "12:58:26"), что в состоянии 2 "Ручной частный" при нажатии кнопки ПЕЧАТЬ автомат A0 вызывается не только с событием e30, но и с событием e40, что некорректно, хотя и не приводит к ошибке в работе модуля.

Указанная ошибка присутствует в файле print_buttons.c и позднее была устранена.

Ниже приведем пример диагностирующего (полного) протокола:

```
12:56:43.668* z51. Урезать файл предварительного просмотра.
12:56:43.708* z55_0. Disable suspending.
12:57:29.045{ A0: в состоянии 0 запущен с событием e40
12:57:29.065> x10 - выбранный для печати файл существует - вернул 1
12:57:29.065T A0: перешел из состояния 0 в состояние 2
12:57:29.085* z800. Показать окно предварительного просмотра.
12:57:29.095* z45. Очистить окно предварительного просмотра.
12:57:29.105* z51. Урезать файл предварительного просмотра.
12:57:29.135* z220. Начать просмотр выбранного файла.
12:57:29.215} A0: завершил обработку события e40 в состоянии 2
12:58:26.911{ A0: в состоянии 2 запущен с событием e40
12:58:26.921} A0: завершил обработку события e40 в состоянии 2
12:58:26.921{ A0: в состоянии 2 запущен с событием e30
12:58:26.931* z200. Добавить выбранный файл в очередь печати.
12:58:26.991} A0: завершил обработку события e30 в состоянии 2
12:58:39.670{ A0: в состоянии 2 запущен с событием e11
12:58:39.680T A0: перешел из состояния 2 в состояние 0
12:58:39.680} A0: завершил обработку события e11 в состоянии 0
12:58:47.829{ A0: в состоянии 0 запущен с событием e30
12:58:47.829* z210. Добавить выбранные фрагменты в очередь печати.
12:58:48.009} A0: завершил обработку события e30 в состоянии 0
12:58:59.099{ A0: в состоянии 0 запущен с событием e12
12:58:59.099T A0: перешел из состояния 0 в состояние 1
12:58:59.109* z260. Напечатать заголовок таблицы событий.
12:58:59.149} A0: завершил обработку события e12 в состоянии 1
12:58:59.149{ A0: в состоянии 1 запущен с событием e110
12:58:59.179> x70 - размер файла печати больше допустимого - вернул 0
12:58:59.199* z500_0. Ответ клиенту - сообщение обработано успешно.
12:58:59.209* z250. Напечатать строку с информацией о событии.
12:58:59.219} A0: завершил обработку события e110 в состоянии 1
12:58:59.239{ A0: в состоянии 1 запущен с событием e110
12:58:59.259> x70 - размер файла печати больше допустимого - вернул 0
12:58:59.279* z500_0. Ответ клиенту - сообщение обработано успешно.
12:58:59.289* z250. Напечатать строку с информацией о событии.
12:58:59.299} A0: завершил обработку события e110 в состоянии 1
12:59:07.518{ A0: в состоянии 1 запущен с событием e13
12:59:07.528> x70 - размер файла печати больше допустимого - вернул 0
12:59:07.528T A0: перешел из состояния 1 в состояние 0
12:59:07.538} A0: завершил обработку события e13 в состоянии 0
```

Приведем текст соответствующего проверяющего (короткого) протокола:

```
13:06:59.886* z51. Урезать файл предварительного просмотра.
13:06:59.926* z55_0. Disable suspending.
13:07:06.055{ A0: в состоянии 0 запущен с событием e12
13:07:06.065* z260. Напечатать заголовок таблицы событий.
13:07:06.125} A0: завершил обработку события e12 в состоянии 1
13:07:06.125{ A0: в состоянии 1 запущен с событием e110
13:07:06.145* z500_0. Ответ клиенту - сообщение обработано успешно.
13:07:06.165* z250. Напечатать строку с информацией о событии.
13:07:06.185} A0: завершил обработку события e110 в состоянии 1
13:07:06.195{ A0: в состоянии 1 запущен с событием e110
13:07:06.215* z500_0. Ответ клиенту - сообщение обработано успешно.
13:07:06.235* z250. Напечатать строку с информацией о событии.
13:07:06.245} A0: завершил обработку события e110 в состоянии 1
13:07:08.805{ A0: в состоянии 1 запущен с событием e13
13:07:08.815} A0: завершил обработку события e13 в состоянии 0
13:07:18.605{ A0: в состоянии 0 запущен с событием e30
13:07:18.605* z210. Добавить выбранные фрагменты в очередь печати.
13:07:18.744} A0: завершил обработку события e30 в состоянии 0
13:07:28.864{ A0: в состоянии 0 запущен с событием e40
13:07:28.874* z800. Показать окно предварительного просмотра.
13:07:28.894* z45. Очистить окно предварительного просмотра.
13:07:28.904* z51. Урезать файл предварительного просмотра.
13:07:28.924* z220. Начать просмотр выбранного файла.
13:07:28.954} A0: завершил обработку события e40 в состоянии 2
13:07:38.053{ A0: в состоянии 2 запущен с событием e40
13:07:38.053} A0: завершил обработку события e40 в состоянии 2
13:07:38.063{ A0: в состоянии 2 запущен с событием e30
13:07:38.073* z200. Добавить выбранный файл в очередь печати.
13:07:38.103} A0: завершил обработку события e30 в состоянии 2
13:07:42.643{ A0: в состоянии 2 запущен с событием e12
13:07:42.643* z260. Напечатать заголовок таблицы событий.
13:07:42.693} A0: завершил обработку события e12 в состоянии 1
13:08:04.601{ A0: в состоянии 1 запущен с событием e13
13:08:04.601} A0: завершил обработку события e13 в состоянии 0
13:08:07.561{ A0: в состоянии 0 запущен с событием e11
13:08:07.571} A0: завершил обработку события e11 в состоянии 0
13:08:09.771{ A0: в состоянии 0 запущен с событием e40
13:08:09.771* z800. Показать окно предварительного просмотра.
13:08:09.781* z45. Очистить окно предварительного просмотра.
13:08:09.791* z51. Урезать файл предварительного просмотра.
13:08:09.841* z220. Начать просмотр выбранного файла.
13:08:09.861} A0: завершил обработку события e40 в состоянии 2
13:08:11.501{ A0: в состоянии 2 запущен с событием e11
13:08:11.511} A0: завершил обработку события e11 в состоянии 0
```

Протоколы функционирования менеджера печати строятся аналогично и, по этой причине, не приводятся.

4.3. Система управления судовым дизель-генератором

Разработанная система предназначена для управления двумя дизель-генераторами, функционирующими по одинаковым алгоритмам. Система управления содержит порядка 50 дискретных входов, 50 аналоговых входов, 50 дискретных выходов, до 20 одновременно активных выдержек времени и 5 видеокадров.

Программы выполнены для операционной системы QNX 4.25 и графической оболочки Photon 1.14.

Для отладки разработанной системы был создан простейший программный имитатор объекта управления (дизель-генератора), также спроектированный с использованием предлагаемой технологии. Фрагмент программной документации этого имитатора приведен в следующем разделе.

4.3.1. Документирование процесса проектирования системы управления

Приведем перечень сокращений, которые используются в дальнейшем:

ГО - газоохладитель, газоотвод;
ГПК - главный пусковой клапан;
Д1СЧВ, Д2СЧВ, Д3СЧВ - датчики 1-ой, 2-ой и 3-ей ступеней частоты вращения;
ДВПМ - датчик валопроворотного механизма;
ДГ - дизель-генератор;
ДПКВ - датчик проворота коленвала;
ДППВ - датчик положения предельного выключателя;
ДЧВ - датчик частоты вращения;
МВП - механизм валопроворотный;
МПН - маслопрокачивающий насос;
ОКС - общекорабельная система;
ПРЕД - предельная частота вращения;
РЧВ - рабочая частота вращения, регулятор частоты вращения;
СУ - система управления;
ЧВНЗ - частота вращения начала заливания;
ЧВО - частота вращения останова.

В настоящей работе не приведены следующие документы, включенные в программную документацию: структурная схема системы управления, отражающая ее приборный состав; видеокадры операторского интерфейса системы; перечни

событий, входных переменных и выходных воздействий; тексты системозависимой части программы.

Взаимодействие разработанных автоматов отражено на схеме (рис. 23), которая напоминает диаграмму классов, создаваемую при объектно-ориентированном проектировании.

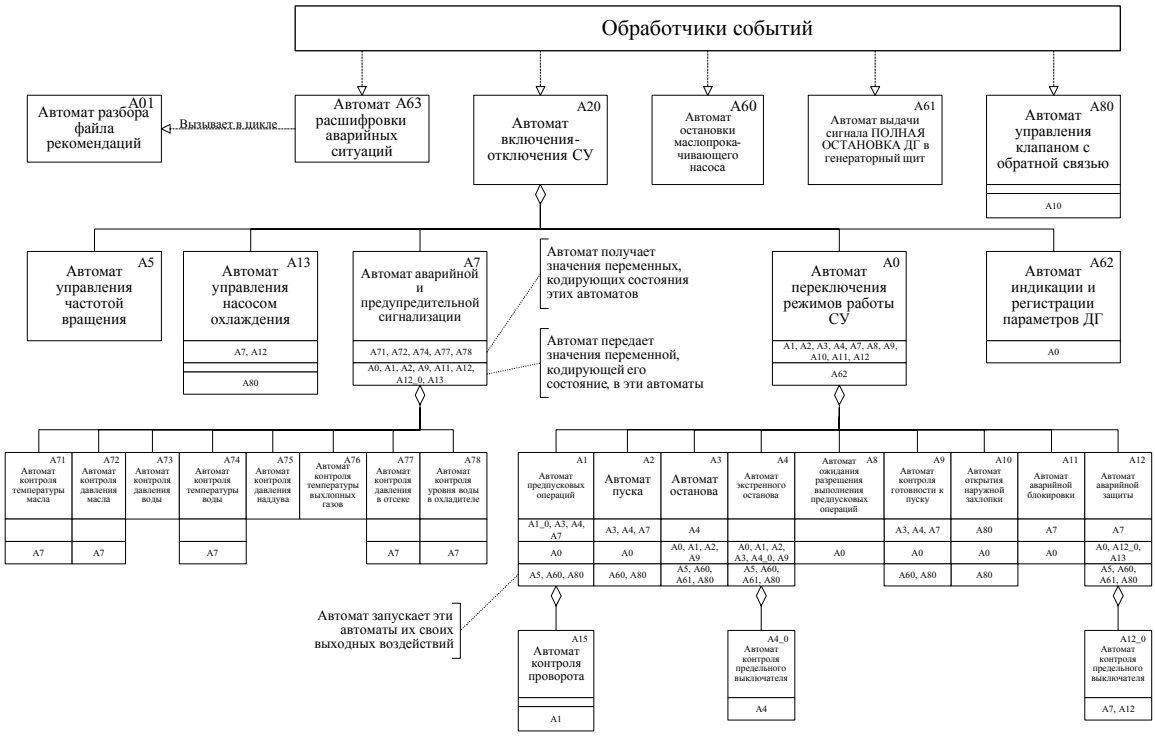

Рис. 23. Схема взаимодействия автоматов

В соответствии с этой схемой разработана остальная программная документация системы управления, включающая тексты программ системонезависимой части и протоколы работы системы, подверждающие правильность ее функционирования. Однако, в силу ее большого объема, она приводится не полностью. Дальнейшее изложение материала проведем на фрагменте системы, схема взаимодействия автоматов которого показана на рис. 24. В этой схеме, также, как и в предыдущей, автоматы взаимодействуют по вложенности, вызываемости и обмену номерами состояний.

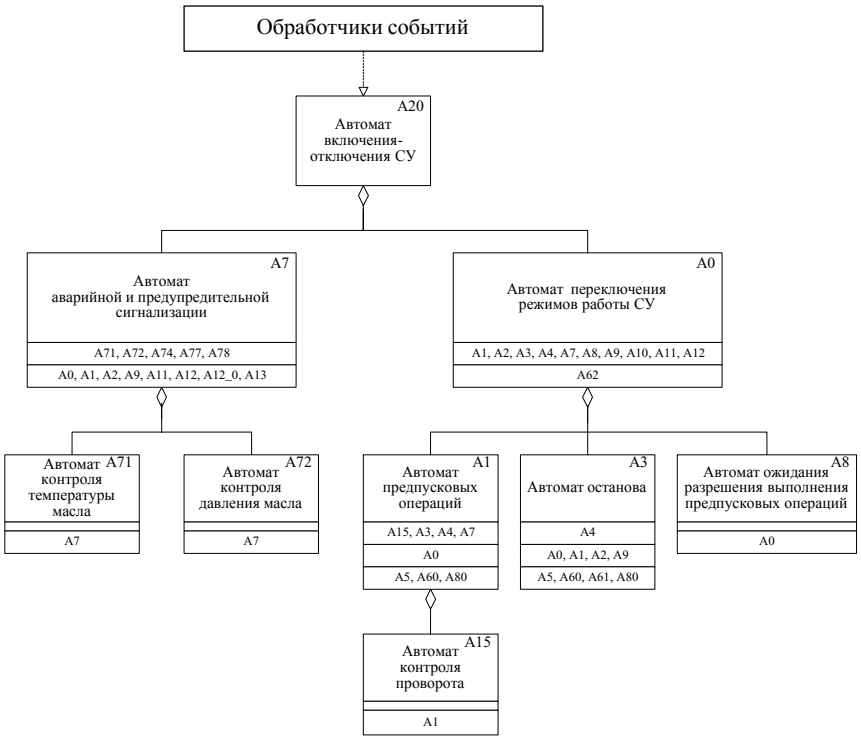

Рис. 24. Фрагмент схемы взаимодействия автоматов

Для каждого из автоматов, приведенных на этой схеме, разработаны четыре документа: словесное описание; схема связей; граф переходов; текст функции, реализующей автомат. При этом для автоматов, реализующих режимы работы, явно описанные в техническом задании, в качестве словесного описания приводятся его соответствующие фрагменты. Выполнение указанными автоматами, построенными "по мотивам" этих описаний, требований технического задания подтверждается протоколами работы системы. Пример одного из таких протоколов приведен в конце раздела 1.

4.3.2. Автомат включения-отключения системы управления

4.3.2.1. *Словесное описание*

Автомат, реализующий данный алгоритм является головным в схеме взаимодействия автоматов. Этот автомат обеспечивает выполнение перечисленных ниже действий и вызов вложенных автоматов.

1. При запуске программы выполняется инициализация всех органов управления.

2. При переводе переключателя "дистанционное/местное управление" в положении "местное" автомат выполняет инициализацию исполнительных механизмов и переходит в отключенное состояние.

3. При переводе переключателя "дистанционное/местное управление" в положении "дистанционное" автомат запускает генератор синхроимпульсов (период 0.5 с) и начинает выполнение алгоритма управления.

При этом:

– при нажатии кнопки "Квитирование" все мигающие табло переводятся в постоянное свечение и отключается обобщенная звуковая сигнализация;

– при появлении сигнала "Предельный выключатель" включается соотоветствующее табло;

– при исчезновении сигнала "Предельный выключатель" выключается соотоветствующее табло.

Схема связей автомата и граф переходов приведены на рис. 25, 26.

4.3.2.2. *Схема связей*

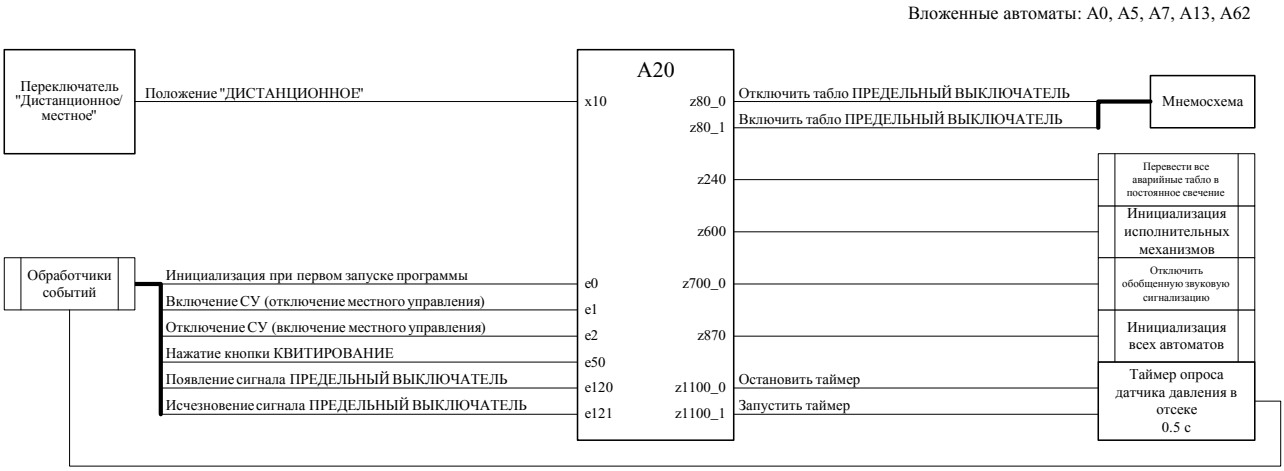

Рис. 25. Схема связей автомата включения-отключения системы управления

4.3.2.3. *Граф переходов*

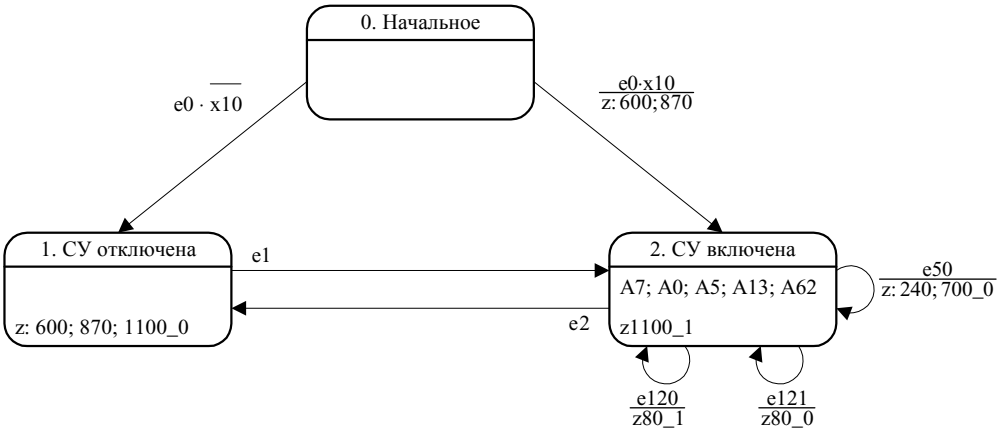

Рис. 26. Граф переходов автомата включения-отключения системы управления

4.3.2.4. *Текст функции, реализующей автомат*

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

void A20( int e, dg_t *dg )
{
  int y_old = dg->y20 ;

  #ifdef HEAD_EVENTS_LOGGING
    log_exec( dg, "A20", y_old, e ) ;
  #endif
```

```
  switch( dg->y20 )
  {
    case 0:
      if( e == 0 && !x10(dg) )                          dg->y20 = 1 ;
      else
      if( e == 0 && x10(dg) ) { z600(dg) ; z870(dg) ;  dg->y20 = 2 ; } ;
    break ;

    case 1:
      if( e == 1 )                                      dg->y20 = 2 ;
    break ;

    case 2:
      A7( e, dg ) ; A0( e, dg ) ; A5( e, dg ) ; A13( e, dg ) ; A62( e, dg ) ;
      if( e == 2 )                                      dg->y20 = 1 ;
      else
      if( e == 50 ) {         z240(dg) ; z700_0(dg) ; }
      else
      if( e == 121 ) {         z80_0(dg) ; }
      else
      if( e == 120 ) {         z80_1(dg) ; } ;
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A20: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == dg->y20 ) goto A20_end ;

  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A20", y_old, dg->y20 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y20 )
  {
    case 1:
      z600(dg) ; z870(dg) ; z1100_0(dg) ;
    break ;

    case 2:
      A7( 0, dg ) ; A0( 0, dg ) ; A5( 0, dg ) ; A13( 0, dg ) ; A62( 0, dg ) ;
      z1100_1(dg) ;
    break ;

  } ;

  A20_end:
  #ifdef HEAD_ENDS_LOGGING
    log_end( dg, "A20", dg->y20, e ) ;
  #endif
  ;
} ;
```

4.3.3. Автомат переключения режимов работы системы управления

4.3.3.1. *Словесное описание*

Автомат переключения режимов работы реализует основной алгоритм управления. Автоматы, вложенные в его

состояния, обеспечивают детализацию алгоритма работы в соответствующих режимах. Перечислим эти режимы.

1. Дизель остановлен. В рассматриваемом режиме осуществляется проверка условий начала выполнения алгоритма предпусковых операций. Если при нахождении системы управления в этом режиме дизель-генератор был запущен в обход ее, то система отреагирует на это переходом в установившийся режим. Срабатывание одного из алгоритмов аварийной и предупредительной сигнализации в этом режиме приводит к переходу системы в режим аварийной блокировки.

2. Предпусковые операции. В рассматриваемом режиме выполняется алгоритм предпусковых операций.

3. К пуску готов. Режим соответствует удачному завершению выполнения алгоритма предпусковых операций. В этом режиме система проверяет сохранились ли условия, разрешающие пуск. При нажатии оператором кнопки "Пуск" начинается выполнение алгоритма пуска.

4. Выполняется пуск. В рассматриваемом режиме выполняется алгоритм пуска, при успешном осуществлении которого система переходит в установившийся режим.

5. Установившийся режим. В этом режиме при необходимости выполняется алгоритм открытия наружной захлопки, реализуемый автоматом A10. Выход из этого режима происходит в начале выполнения одного из видов останова, при срабатывании алгоритма аварийной защиты или при останове дизель-генератора в обход системы управления.

6. Останов. В рассматриваемом режиме выполненяется алгоритм останова. Указанный алгоритм может быть прерван в случае экстренного останова.

7. Экстренный останов. В рассматриваемом режиме выполненяется алгоритм экстренного останова. После

завершения этого алгоритма система переходит в режим аварийной блокировки.

8. Аварийная блокировка. В этом режиме система ждет пока оператор подтвердит свою реакцию на аварию нажатием кнопки "Квитирование" и, устранив аварию, нажмет кнопку "Разблокировка".

9. Аварийная защита. Этот режим аналогичен режиму экстренного останова.

Схема связей автомата и граф переходов приведены на рис. 27, 28.

4.3.3.2. *Схема связей*

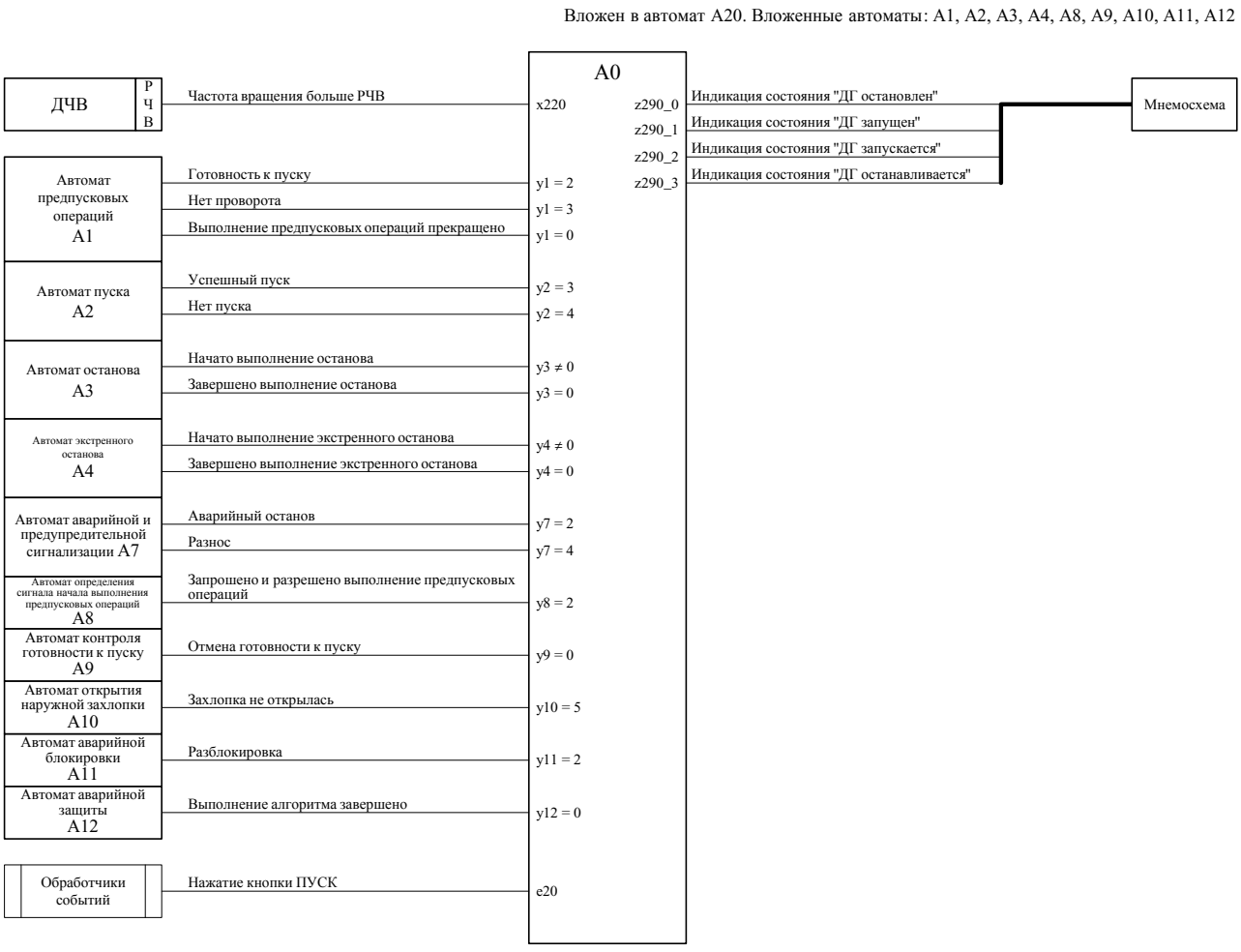

Рис. 27. Схема связей автомата переключения режимов работы системы управления

4.3.3.3. *Граф переходов*

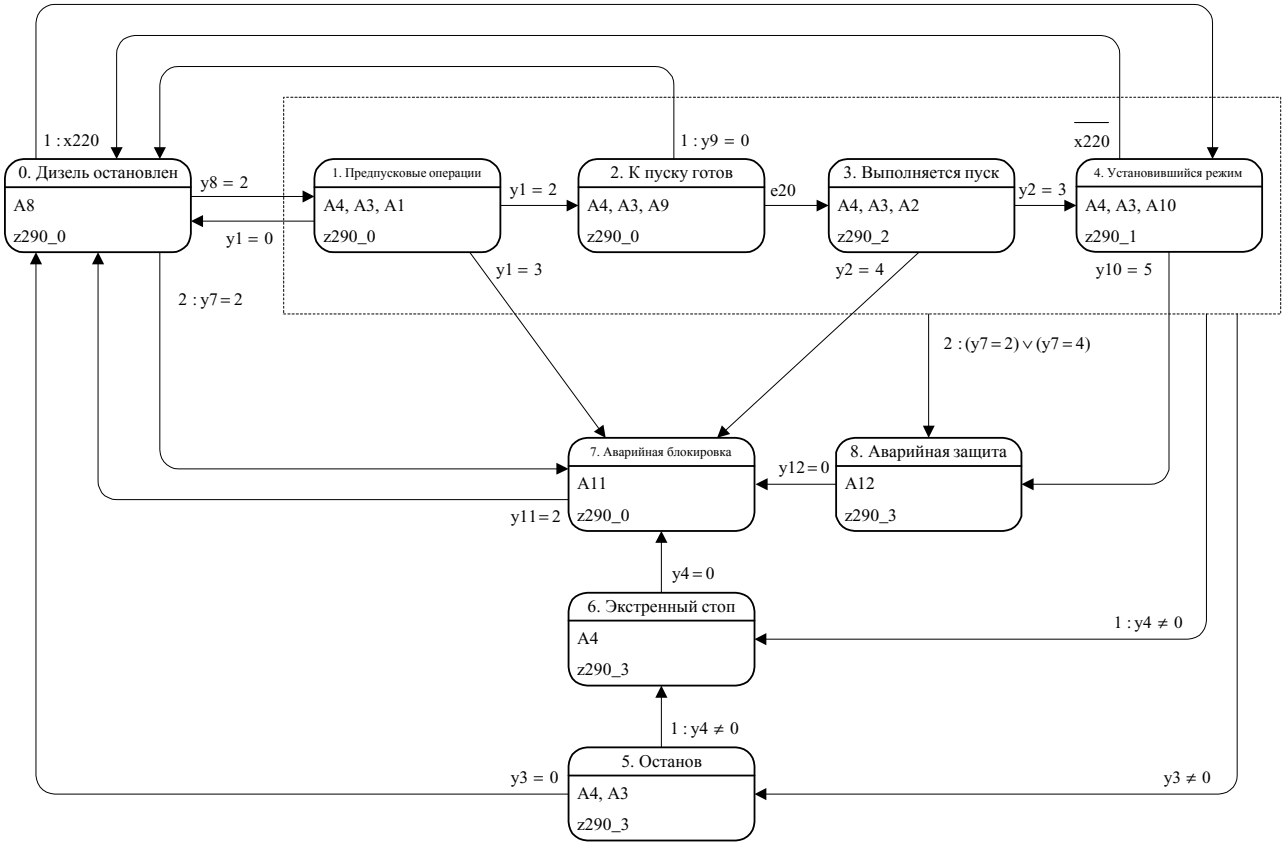

Рис. 28. Граф переходов автомата переключения режимов работы системы управления

4.3.3.4. *Текст функции, реализующей автомат*

```
#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
} ;
```

4.3.4. Автомат предпусковых операций

4.3.4.1. *Словесное описание*

1. Перечислим сигналы, блокирующие проведение предпусковых операций.

1.1. Обобщенный сигнал "Блокировка пуска" из СУ ОКС. При наличии этого сигнала выдается сигнал "Подготовка пуска" в указанную систему.

1.2. Сигнал "Механизм валопроворотный не отключен".

1.3. Сигнал "Предельный выключатель".

1.4. Дизель не остановлен — частота вращения превышает рабочую частоту.

1.5. Команда на выполнение любого вида останова.

1.6. Наличие сигнала "Вода в охладителе" (аварийный уровень).

2. При наличии хотя бы одного из сигналов, указанных в п.п.1.2...1.6, система управления не должна принимать управляющую команду подготовки к пуску.

3. При нажатии оператором кнопки "Подготовка к пуску" выдается сигнал в СУ ОКС. После снятия сигнала "Блокировка пуска" из СУ ОКС и при отсутствии блокирующих сигналов по п.п. 1.2...1.6, система должна выполнить перечисленные ниже действия.

3.1. Запомнить команду на подготовку к пуску.

3.2. Включить табло "Подготовка к пуску".

3.3. Выдать команду на включение маслопрокачивающего насоса, замкнув контакт в цепи его магнитного пускателя.

3.4. Включить табло "Прокачка маслом" при замыкании контакта пускателя маслопрокачивающего насоса.

3.5. Подать питание на электромагнит стопа регулятора частоты вращения.

3.6. Подать команду на открытие клапана подачи воздуха к дизель-генератору.

3.7. Подать команду на открытие клапана воздуха низкого давления.

3.8. Включить табло "Проворот".

3.9. Включить контроль времени проворота (60 с).

3.10. Выдать команду в регулятор частоты вращения на установку первой ступени частоты, и через 4 с снять эту команду.

4. После получения 3-х импульсов от датчика проворота коленчатого вала, до истечения времени по п. 3.9, система управления должна выполнить перечисленные ниже действия.

4.1. Снять питание с электромагнита клапана воздуха низкого давления.

4.2. Отключить табло "Проворот".

4.3. Отключить контроль времени проворота (60 с).

5. При превышении давлением масла предпускового давления и отработки команды на уменьшение заданной частоты вращения до первой ступени, система управления должна выполнить перечисленные ниже действия.

5.1. Снять команду на включение маслопрокачивающего насоса.

5.2. Снять команды на подготовку к пуску, перечисленные в п.3.

5.3. Отключить табло "Подготовка к пуску".

5.4. Включить табло "Пуск разрешен".

5.5. Включить память завершения предпусковых операций и дать разрешение на выполнение пуска.

6. В случае снижения давления масла ниже предпускового давления или увеличения затяжки пружины регулятора частоты вращения до третьей позиции или появления блокирующих сигналов по п. 1.1—1.3, 1.5, 1.6 система управления должна выполнить перечисленные ниже действия.

6.1. Отключить память завершения предпусковых операций.

6.2. Отключить табло "Пуск разрешен".

6.3. Разомкнуть контакт остановки маслопрокачивающего насоса и через 4 с вновь замкнуть его.

6.4. Снять питание с электромагнита остановки регулятора частоты вращения.

6.5. Отключить табло "Прокачка маслом" при размыкании контакта пускателя маслопрокачивающего насоса.

6.6. Снять команду на открытие клапана подачи воздуха к дизель-генератору.

7. Если по истечении времени (п. 3.9) от датчика проворота коленвала не поступило 3-х импульсов, то система управления должна выполнить перечисленные ниже действия.

7.1. Включить аварийное табло "Нет проворота".

7.2. Включить обобщенный сигнал звуковой сигнализации.

7.3. Отключить табло "Подготовка к пуску".

7.4. Снять питание с электромагнита клапана пускового воздуха низкого давления.

7.5. Отключить контроль времени проворота (60 сек).

7.6. Снять команду на включение маслопрокачивающего насоса.

7.7. Разомкнуть контакт остановки маслопрокачивающего насоса и через 4 с вновь замкнуть этот контакт.

7.8. Отключить табло "Прокачка маслом" при размыкании контакта пускателя маслопрокачивающего насоса.

7.9. Отключить память команды на подготовку к пуску.

7.10. Снять команду на открытие клапана подачи воздуха к дизель-генератору.

8. Отключение обобщенной звуковой сигнализации и перевод аварийного табло "Нет проворота" в постоянное свечение производится нажатием кнопки "Квитирование".

9. Разблокировка системы управления и отключение табло "Нет проворота" производится нажатием кнопки "Разблокировка".

Схема связей автомата и граф переходов приведены на рис. 29, 30.

4.3.4.2. *Схема связей*

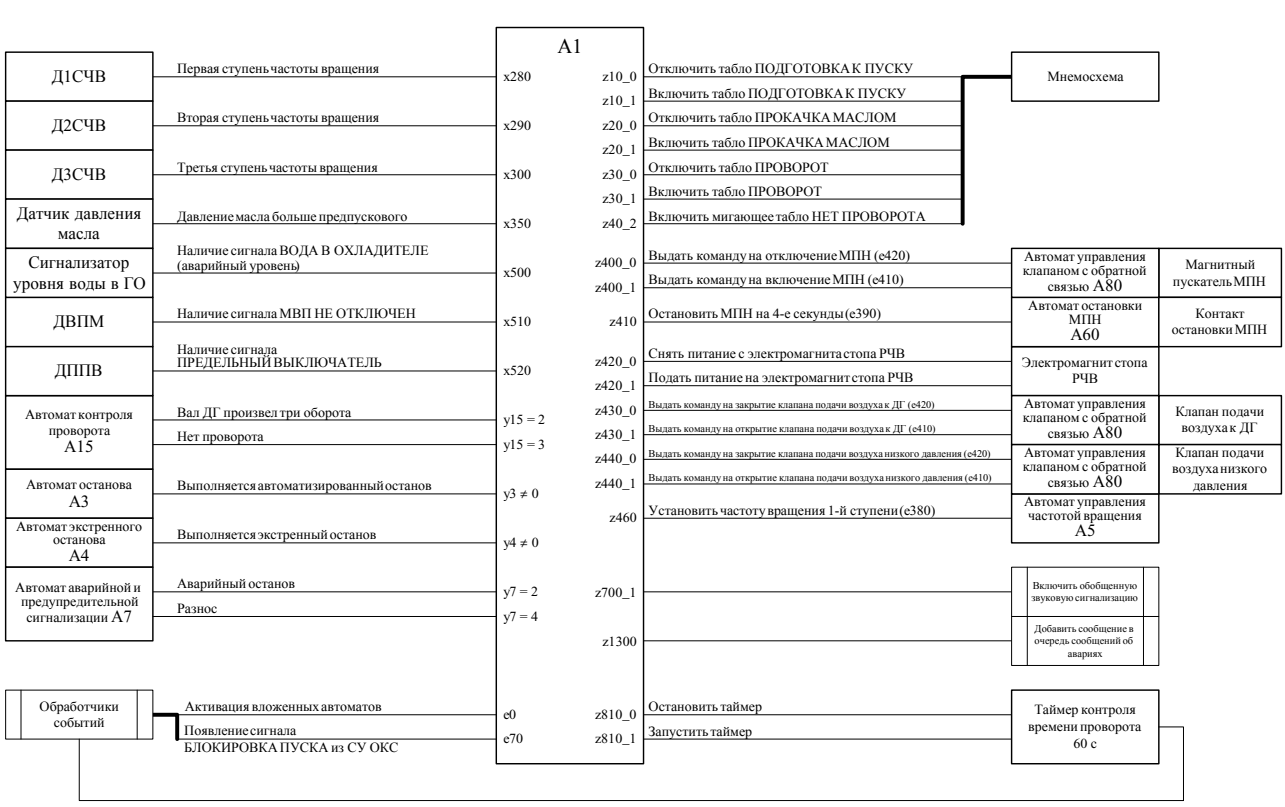

Рис. 29. Схема связей автомата предпусковых операций

4.3.4.3. *Граф переходов*

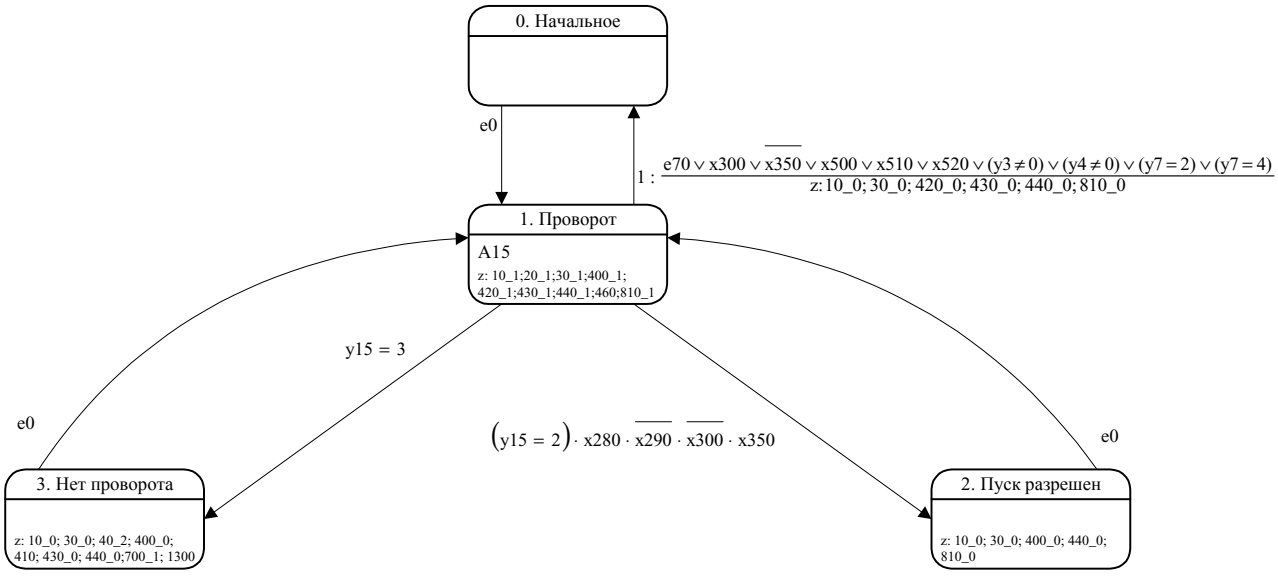

Рис. 30. Граф переходов автомата предпусковых операций

4.3.4.4. *Текст функции, реализующей автомат*

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

void A1( int e, dg_t *dg )
{
  int y_old = dg->y1 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A1", y_old, e ) ;
  #endif

  switch( dg->y1 )
  {
    case 0:
      if( e == 0 )
                                                dg->y1 = 1 ;
    break ;

    case 1:
      A1_0( e, dg ) ;
      if(    e == 70 || x300(dg) || !x350(dg) || x500(dg) || x510(dg) || x520(dg)
          || dg->y3 != 0 || dg->y4 != 0 || dg->y7 == 2 || dg->y7 == 4 )
      { z10_0(dg) ; z30_0(dg) ; z420_0(dg) ; z430_0(dg) ; z440_0(dg) ; z810_0(dg) ;
                                                dg->y1 = 0 ; }
      else
      if( dg->y1_0 == 3 )
                                                dg->y1 = 3 ;
      else
      if( dg->y1_0 == 2 && x350(dg) && x280(dg) && !x290(dg) && !x300(dg) )
                                                dg->y1 = 2 ;
    break ;

    case 2:
      if( e == 0 )
                                                dg->y1 = 1 ;
    break ;

    case 3:
      if( e == 0 )
                                                dg->y1 = 1 ;
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A1: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == dg->y1 ) goto A1_end ;

  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A1", y_old, dg->y1 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y1 )
  {
    case 1:
      A1_0( 0, dg ) ;
      z10_1(dg) ; z20_1(dg) ; z30_1(dg) ; z400_1(dg) ; z420_1(dg) ;
      z430_1(dg) ; z440_1(dg) ; z460(dg) ; z810_1(dg) ;
    break ;
```

```
    case 2:
      z10_0(dg) ; z30_0(dg) ; z400_0(dg) ; z440_0(dg) ; z810_0(dg) ;
    break ;

    case 3:
      z10_0(dg) ; z30_0(dg) ; z40_2(dg) ; z400_0(dg) ; z430_0(dg) ;
      z440_0(dg) ; z410(dg) ;
      z700_1(dg) ; z1300(dg, MSG_NO_TURN) ;
    break ;
  } ;

  A1_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A1", dg->y1, e ) ;
  #endif
  ;
} ;
```

4.3.5. Автомат контроля проворота

4.3.5.1. *Словесное описание*

Автомат контроля проворота реализует часть алгоритма предпусковых операций, который описан в предыдущем подразделе (например, в п.п. 4, 7).

Обратим внимание, что в качестве объекта управления этого автомата выступает не "физический" объект, а вычислительное устройство, реализуемое программно.

Схема связей автомата и граф переходов приведены на рис. 31, 32.

4.3.5.2. *Схема связей*

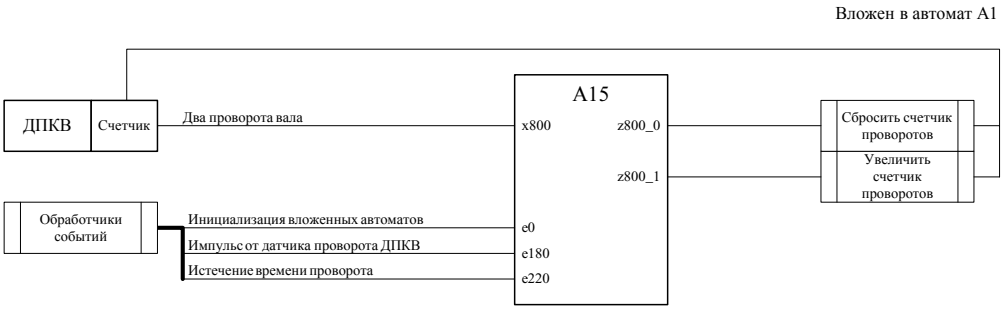

Рис. 31. Схема связей автомата контроля проворота

4.3.5.3. *Граф переходов*

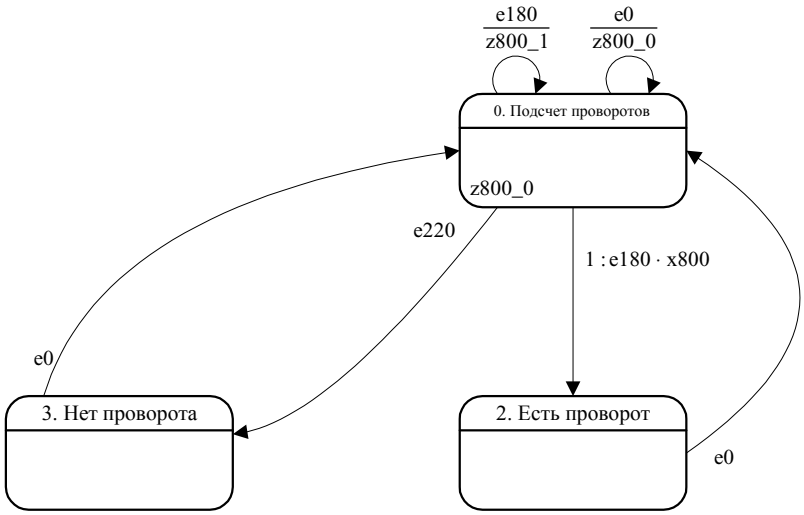

Рис. 32. Граф переходов автомата контроля проворота

4.3.5.4. *Текст функции, реализующей автомат*

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

void A15( int e, dg_t *dg )
{
  int y_old = dg->y15 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A15", y_old, e ) ;
  #endif

  switch( dg->y15 )
  {
    case 0:
      if( e == 180 && x800(dg) )             dg->y15 = 2 ;
      else
      if( e == 220 )                         dg->y15 = 3 ;
      else
      if( e == 0 )   {     z800_0(dg) ; }
      else
      if( e == 180 ) {     z800_1(dg) ; } ;
    break ;

    case 2:
      if( e == 0 )                           dg->y15 = 0 ;
    break ;

    case 3:
      if( e == 0 )                           dg->y15 = 0 ;
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A15: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == dg->y15 ) goto A15_end ;
```

```
  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A15", y_old, dg->y15 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y15 )
  {
    case 0:
      z800_0(dg) ;
    break ;
  } ;

  A15_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A15", dg->y15, e ) ;
  #endif
  ;
} ;
```

4.3.6. Автомат останова

4.3.6.1. *Словесное описание*

1. При нажатии оператором кнопки "Останов" система управления должна выполнить перечисленные ниже действия.

1.1. Отключить память команды на подготовку к пуску и табло "Подготовка к пуску", память готовности к пуску и табло "Пуск разрешен", память команды пуск и табло "Пуск", если соответствующие операции в момент подачи команды выполнялись.

1.2. Включить память команды останов.

1.3. Включить табло "Останов".

1.4. Подать питание на электромагнит остановки регулятора частоты вращения.

1.5. Выдать команду на уменьшение затяжки пружины регулятора частоты вращения до величины соответствующей первой позиции и через 4 с снять эту команду.

2. При уменьшении частоты вращения ниже частоты "начала заливания" система управления должна выдать в СУ ОКС сигнал "Закрыть наружную захлопку".

3. При уменьшении частоты вращения ниже рабочей частоты система управления должна выполнить перечисленные ниже действия.

3.1. Выдать команду на включение маслопрокачивающего насоса.

3.2. Включить отсчет времени прокачки маслом (60 сек).

3.3. Включить табло "Прокачка маслом".

4. При уменьшении частоты вращения ниже частоты вращения останова система управления должна выполнить перечисленные ниже действия.

4.1. Выдать в СУ ОКС сигнал "Полная остановка ДГ".

4.2. Выдать в генераторный щит сигнал "Полная остановка ДГ" длительностью 3 с.

4.3. Выдать команду на закрытие клапана подачи воды в газоохладитель и на закрытие клапана слива воды из сепаратора газоотвода.

5. Через установленное время (п. 3.2) после запуска маслопрокачивающего насоса система управления должна выполнить перечисленные ниже действия.

5.1. Снять команду на включение маслопрокачивающего насоса.

5.2. Разомкнуть контакт остановки маслопрокачивающего насоса и через 4 с вновь замкнуть этот контакт.

5.3. Отключить табло "Прокачка маслом" при размыкании контакта пускателя маслопрокачивающего насоса.

5.4. Снять питание с электромагнита остановки регулятора частоты вращения.

5.5. Отключить память команды останов.

5.6. Отключить табло "Останов".

Схема связей автомата и граф переходов приведены на рис. 33, 34.

4.3.6.2. *Схема связей*

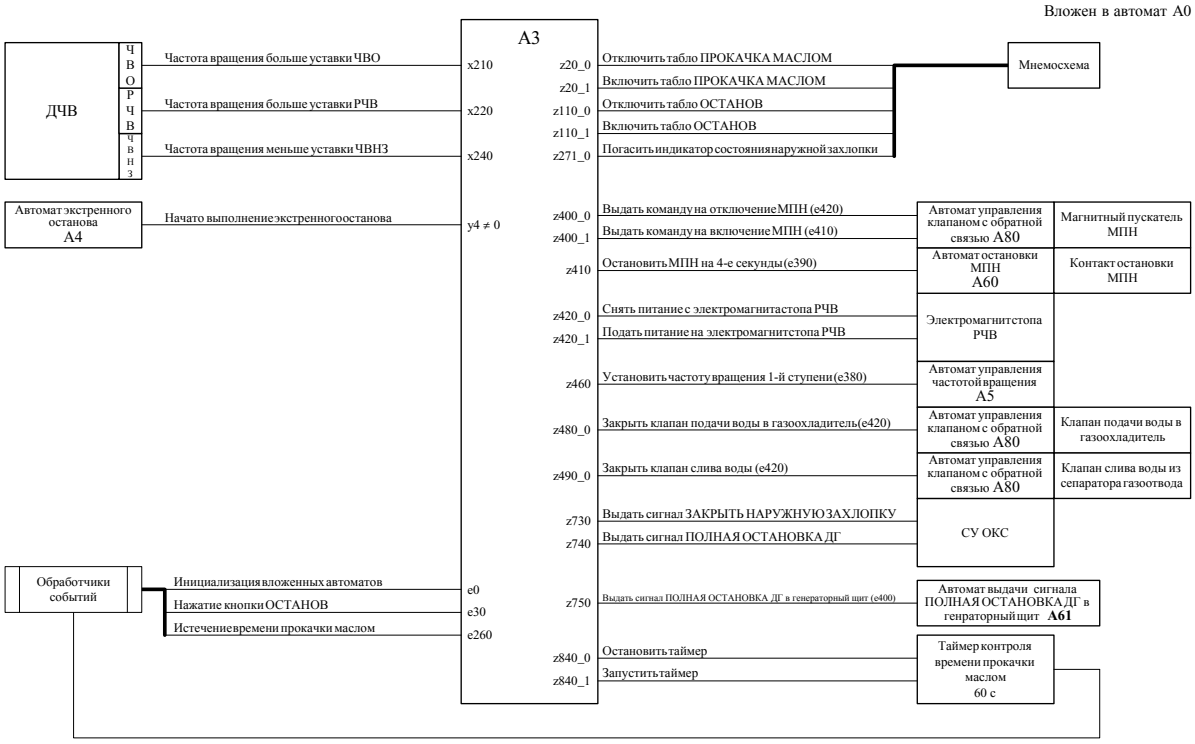

Рис. 33. Схема связей автомата останова

4.3.6.3. *Граф переходов*

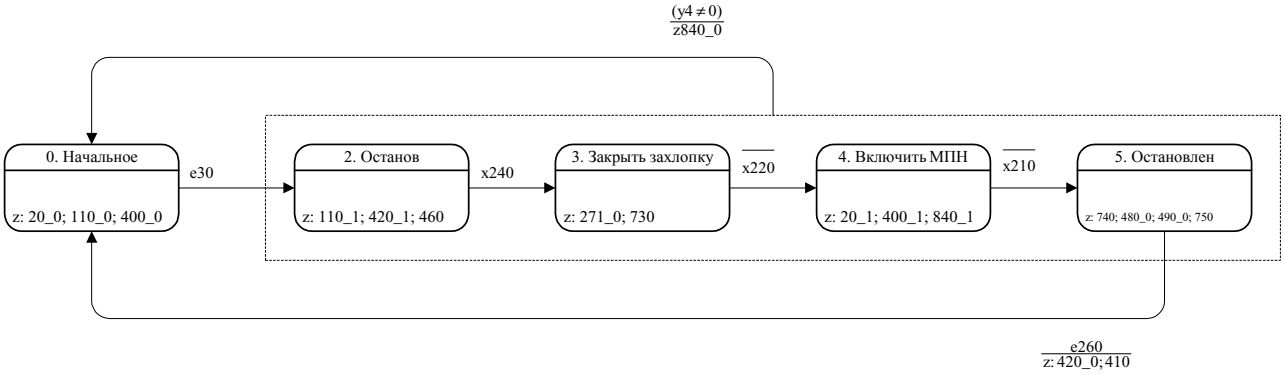

Рис. 34. Граф переходов автомата останова

4.3.6.4. *Текст функции, реализующей автомат*

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

void A3( int e, dg_t *dg )
{
  int y_old = dg->y3 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A3", y_old, e ) ;
  #endif
```

```
switch( dg->y3 )
{
  case 0:
    if( e == 30 )                              dg->y3 = 2 ;
  break ;

  case 2:
    if( dg->y4 != 0 ) { z840_0(dg) ;           dg->y3 = 0 ; }
    else
    if( x240(dg) )                             dg->y3 = 3 ;
  break ;

  case 3:
    if( dg->y4 != 0 ) { z840_0(dg) ;           dg->y3 = 0 ; }
    else
    if( !x220(dg) )                            dg->y3 = 4 ;
  break ;

  case 4:
    if( dg->y4 != 0 ) { z840_0(dg) ;           dg->y3 = 0 ; }
    else
    if( !x210(dg) )                            dg->y3 = 5 ;
  break ;

  case 5:
    if( dg->y4 != 0 ) { z840_0(dg) ;           dg->y3 = 0 ; }
    else
    if( e == 260 ) { z410(dg) ; z420_0(dg) ; dg->y3 = 0 ; }
  break ;

  default:
    #ifdef GRAPH_ERRORS_LOGGING
      log_write( LOG_GRAPH_ERROR, dg->number,
                 "ERROR IN A3: unknown state number!", 0 ) ;
    #endif
  break ;
} ;

if( y_old == dg->y3 ) goto A3_end ;

{
  #ifdef GRAPH_TRANS_LOGGING
    log_trans( dg, "A3", y_old, dg->y3 ) ;
  #endif

  #ifdef DEBUG_FRAME
    update_debug() ;
  #endif
} ;

switch( dg->y3 )
{
  case 0:
    z20_0(dg) ; z110_0(dg) ; z400_0(dg) ;
  break ;

  case 2:
    z110_1(dg) ; z420_1(dg) ; z460(dg) ;
  break ;

  case 3:
    z271_0(dg) ; z730(dg) ;
  break ;

  case 4:
    z20_1(dg) ; z400_1(dg) ; z840_1(dg) ;
  break ;

  case 5:
    z480_0(dg) ; z490_0(dg) ; z740(dg) ; z750(dg) ;
  break ;
} ;
```

```
  A3_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A3", dg->y3, e ) ;
  #endif
  ;
} ;
```

4.3.7. Автомат ожидания разрешения выполнения предпусковых операций

4.3.7.1. *Словесное описание*

Автомат реализует часть алгоритма предпусковых операций, описанного в подразделе 4.3.4. При нажатии кнопки "Подготовка к пуску" автомат проверяет значения сигналов, запрещающих выполнение алгоритма предпусковых операций и, при необходимости, выдает в СУ ОКС сигнал "Подготовка к пуску".

Схема связей автомата и граф переходов приведены на рис. 35, 36.

4.3.7.2. *Схема связей*

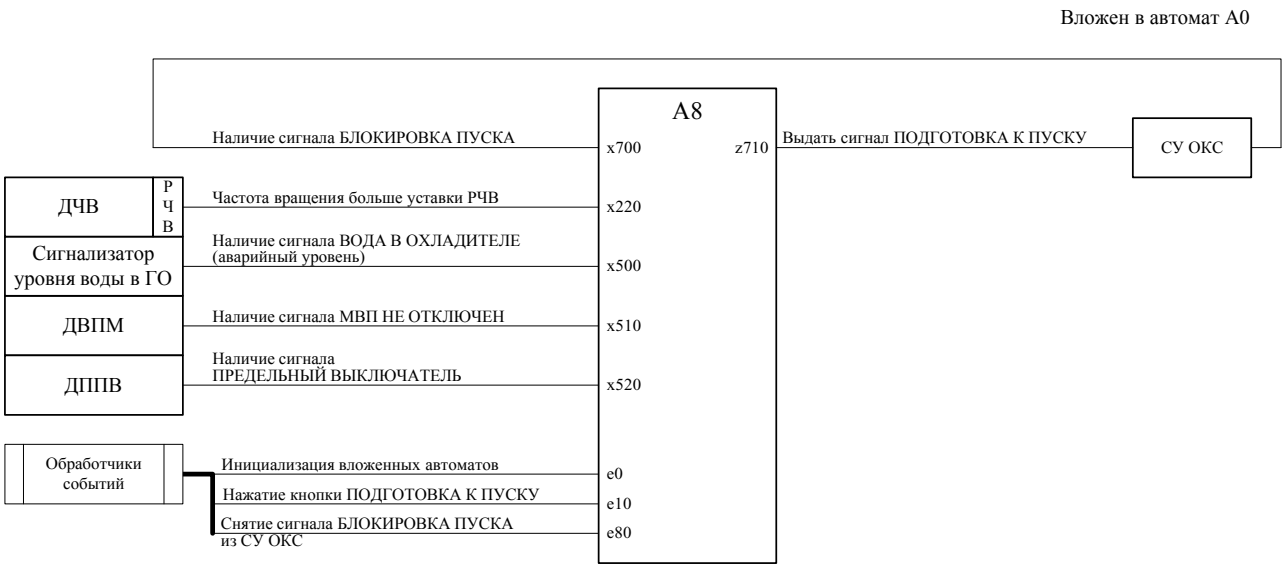

Рис. 35. Схема связей автомата ожидания разрешения выполнения предпусковых операций

4.3.7.3. *Граф переходов*

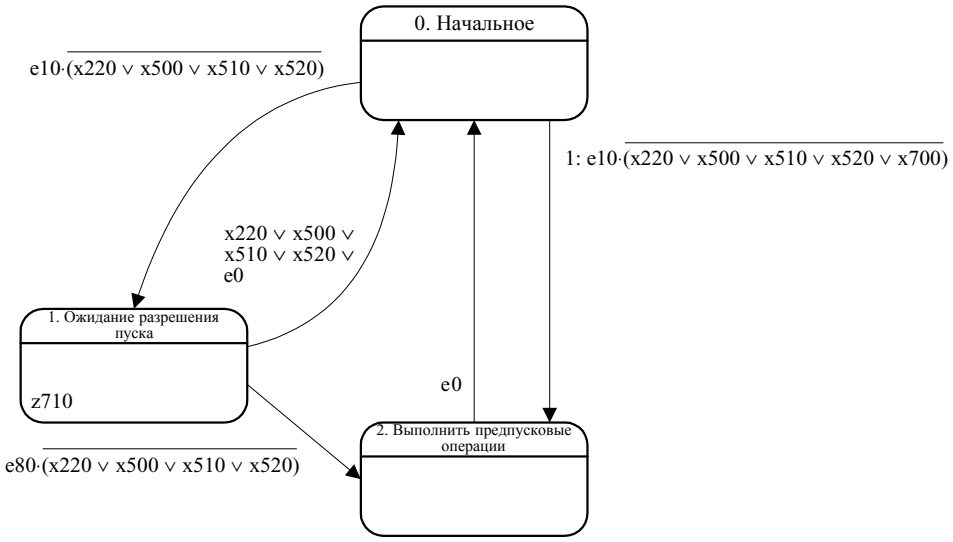

Рис. 36. Граф переходов автомата ожидания разрешения выполнения предпусковых операций

4.3.7.4. *Текст функции, реализующей автомат*

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

void A8( int e, dg_t *dg )
{
  int y_old = dg->y8 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A8", y_old, e ) ;
  #endif

  switch( dg->y8 )
  {
    case 0:
      if( e == 10 && !( x220(dg) || x500(dg) || x510(dg) || x520(dg) || x700(dg)))
                                                dg->y8 = 2 ;
      else
      if( e == 10 && !( x220(dg) || x500(dg) || x510(dg) || x520(dg) ))
                                                dg->y8 = 1 ;
    break ;

    case 1:
      if( x220(dg) || x500(dg) || x510(dg) || x520(dg) || e == 0 )
                                                dg->y8 = 0 ;
      else
      if( e == 80 && !( x220(dg) || x500(dg) || x510(dg) || x520(dg)))
                                                dg->y8 = 2 ;
    break ;

    case 2:
      if( e == 0 )
                                                dg->y8 = 0 ;
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A8: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;
```

```
  if( y_old == dg->y8 ) goto A8_end ;

  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A8", y_old, dg->y8 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y8 )
  {
    case 1:
      z710(dg) ;
    break ;
  } ;

  A8_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A8", dg->y8, e ) ;
  #endif
  ;
} ;
```

4.3.8. Автомат аварийной и предупредительной сигнализации

4.3.8.1. *Словесное описание*

Автомат обобщает работу автоматов контроля параметров дизель-генратора. Выделены четыре режима работы алгоритма аварийной и предупредительной сигнализации.

1. Рабочий режим. Все параметры в допуске.

2. Аварийный останов. Аварийное отклонение как минимум одного из контролируемых параметров.

3. Авария без останова. Аварийное отклонение температуры масла или температуры воды при отключенной защите по этим параметрам.

4. Разнос. Частота вращения дизель-генератора превысила предельно допустимую.

Схема связей автомата и граф переходов приведены на рис. 37, 38.

4.3.8.2. *Схема связей*

Вложен в автомат A20. Вложенные автоматы: A71, A72, A73, A74, A75, A76, A77, A78

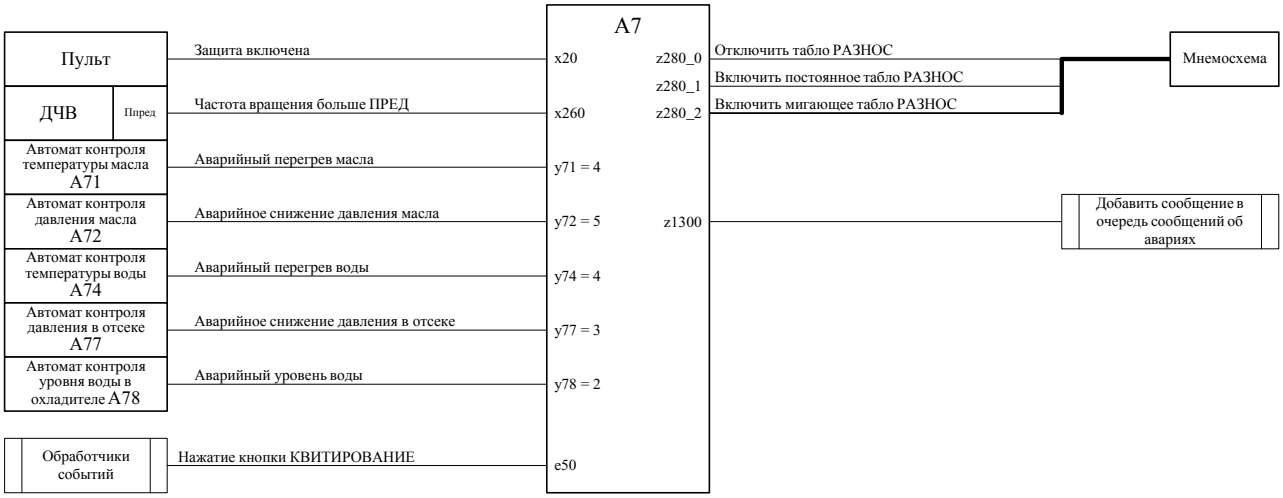

Рис. 37. Схема связей автомата аварийной и предупредительной сигнализации

4.3.8.3. *Граф переходов*

$$C7\_1 = x20 \cdot \big((y71 = 4) \vee (y74 = 4)\big) \vee (y72 = 5) \vee (y77 = 3) \vee (y78 = 2)$$
$$C7\_2 = \overline{x20} \cdot \big((y71 = 4) \vee (y74 = 4)\big)$$

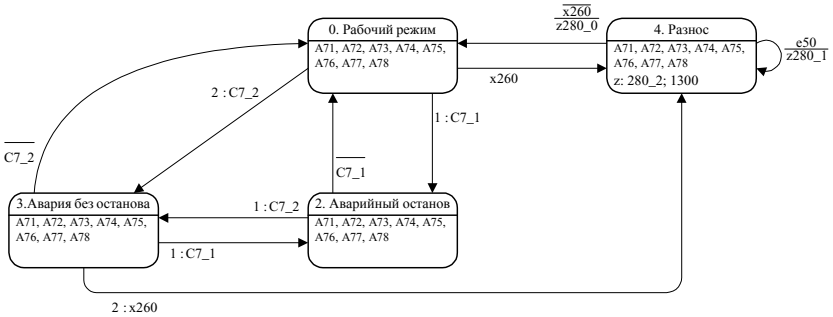

Рис. 38. Граф переходов автомата аварийной и предупредительной сигнализации

4.3.8.4. *Текст функции, реализующей автомат*

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

int C7_1( dg_t *dg )
{
  return    ( x20(dg) && ( dg->y71 == 4 || dg->y74 == 4 ))
         || dg->y72 == 5 || dg->y77 == 3 || dg->y78 == 2 ;
} ;
```

```
int C7_2( dg_t *dg )
{
  return !x20(dg) && ( dg->y71 == 4 || dg->y74 == 4 ) ;
} ;

void A7( int e, dg_t *dg )
{
  int y_old = dg->y7 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A7", y_old, e ) ;
  #endif

  switch( dg->y7 )
  {
    case 0:
      A71(e,dg) ; A72(e,dg) ; A73(e,dg) ; A74(e,dg) ; A75(e,dg) ;
      A76(e,dg) ; A77(e,dg) ; A78(e,dg) ;
      if( C7_1(dg) )                dg->y7 = 2 ;
      else
      if( C7_2(dg) )                dg->y7 = 3 ;
      else
      if( x260(dg) )                dg->y7 = 4 ;
    break ;

    case 2:
      A71(e,dg) ; A72(e,dg) ; A73(e,dg) ; A74(e,dg) ; A75(e,dg) ;
      A76(e,dg) ; A77(e,dg) ; A78(e,dg) ;
      if( C7_2(dg) )                dg->y7 = 3 ;
      else
      if( !C7_1(dg) )               dg->y7 = 0 ;
    break ;

    case 3:
      A71(e,dg) ; A72(e,dg) ; A73(e,dg) ; A74(e,dg) ; A75(e,dg) ;
      A76(e,dg) ; A77(e,dg) ; A78(e,dg) ;
      if( C7_1(dg) )                dg->y7 = 2 ;
      else
      if( x260(dg) )                dg->y7 = 4 ;
      else
      if( !C7_2(dg) )               dg->y7 = 0 ;
    break ;

    case 4:
      A71(e,dg) ; A72(e,dg) ; A73(e,dg) ; A74(e,dg) ; A75(e,dg) ;
      A76(e,dg) ; A77(e,dg) ; A78(e,dg) ;
      if( !x260(dg) ) { z280_0(dg) ; dg->y7 = 0 ; }
      else
      if( e == 50 ) { z280_1(dg) ; }
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A7: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == dg->y7 ) goto A7_end ;

  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A7", y_old, dg->y7 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y7 )
  {
    case 0:
      A71(0,dg) ; A72(0,dg) ; A73(0,dg) ; A74(0,dg) ; A75(0,dg) ;
      A76(0,dg) ; A77(0,dg) ; A78(0,dg) ;
    break ;
```

```
    case 2:
      A71(0,dg) ; A72(0,dg) ; A73(0,dg) ; A74(0,dg) ; A75(0,dg) ;
      A76(0,dg) ; A77(0,dg) ; A78(0,dg) ;
    break ;

    case 3:
      A71(0,dg) ; A72(0,dg) ; A73(0,dg) ; A74(0,dg) ; A75(0,dg) ;
      A76(0,dg) ; A77(0,dg) ; A78(0,dg) ;
    break ;

    case 4:
      A71(0,dg) ; A72(0,dg) ; A73(0,dg) ; A74(0,dg) ; A75(0,dg) ;
      A76(0,dg) ; A77(0,dg) ; A78(0,dg) ;
      z280_2(dg) ; z1300(dg, MSG_RAZNOS_FAIL) ;
    break ;
  } ;

  A7_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A7", dg->y7, e ) ;
  #endif
  ;
} ;
```

4.3.9. Автомат контроля температуры масла

4.3.9.1. *Словесное описание*

Автомат производит контроль температуры масла, определяет и выполняет индикацию следующих ситуаций:

– низкая температура. Температура масла ниже уставки Тмм;

– предупредительный перегрев. Температура масла выше уставки Тмпр;

– аварийный перегрев. Температура масла выше уставки Тма.

Схема связей автомата и граф переходов приведены на рис. 39, 40.

4.3.9.2. *Схема связей*

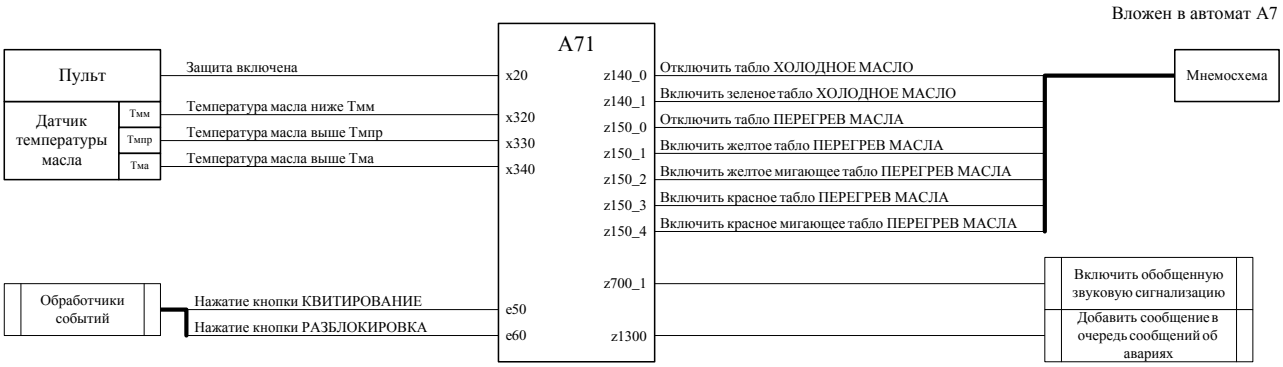

Рис. 39. Схема связей автомата контроля температуры масла

4.3.9.3. *Граф переходов*

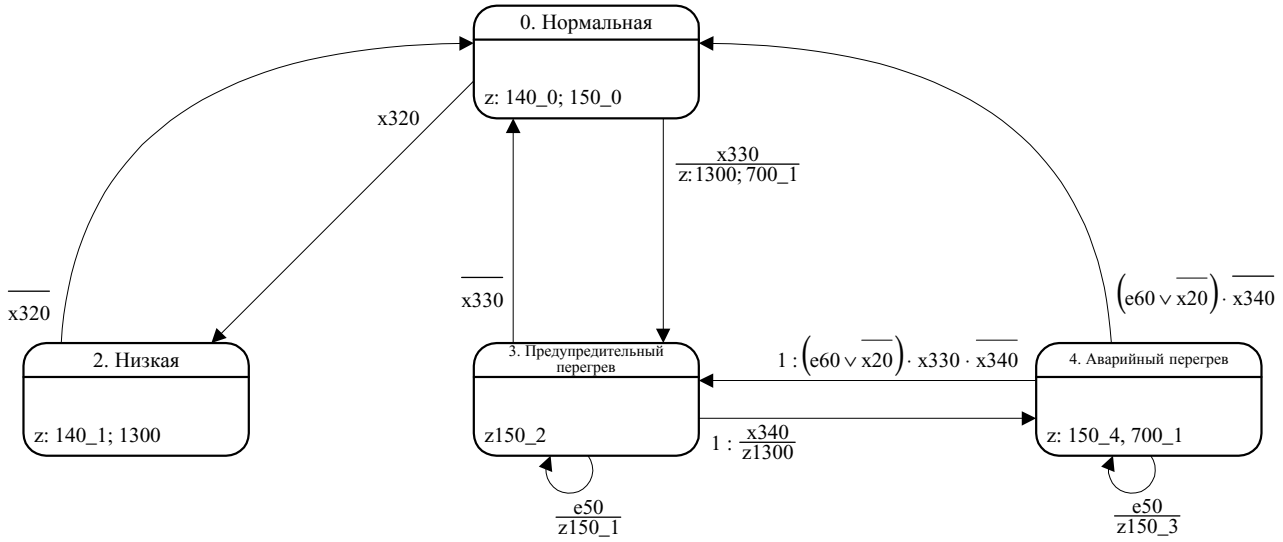

Рис. 40. Граф переходов автомата контроля температуры масла

4.3.9.4. *Текст функции, реализующей автомат*

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

void A71( int e, dg_t *dg )
{
  int y_old = dg->y71 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A71", y_old, e ) ;
  #endif
```

```
  switch( dg->y71 )
  {
    case 0:
      if( x320(dg) )                                dg->y71 = 2 ;
      else
      if( x330(dg) ) { z1300(dg, MSG_OIL_TEMP_HIGH_ALARM) ; z700_1(dg) ;
                                                    dg->y71 = 3 ; }
    break ;

    case 2:
      if( !x320(dg) )                               dg->y71 = 0 ;
    break ;

    case 3:
      if( x340(dg) ) { z1300(dg, MSG_OIL_TEMP_HIGH_FAIL) ;
                                                    dg->y71 = 4 ; }
      else
      if( !x330(dg) )                               dg->y71 = 0 ;
      else
      if( e == 50 ) { z150_1(dg) ; }
    break ;

    case 4:
      if( ( e == 60 || !x20(dg) ) && x330(dg) && !x340(dg))
                                                    dg->y71 = 3 ;
      else
      if( ( e == 60 || !x20(dg) ) && !x340(dg) )
                                                    dg->y71 = 0 ;
      else
      if( e == 50 ) { z150_3(dg) ; }
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A71: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == dg->y71 ) goto A71_end ;

  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A71", y_old, dg->y71 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y71 )
  {
    case 0:
      z140_0(dg) ; z150_0(dg) ;
    break ;

    case 2:
      z140_1(dg) ; z1300(dg, MSG_OIL_TEMP_LOW_ALARM) ;
    break ;

    case 3:
      z150_2(dg) ;
    break ;

    case 4:
      z150_4(dg) ; z700_1(dg) ;
    break ;
  } ;

  A71_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A71", dg->y71, e ) ;
  #endif
  ;
} ;
```

4.3.10. Автомат контроля давления масла

4.3.10.1. *Словесное описание*

Автомат производит контроль давления масла, определяет и выполняет индикацию следующих ситуаций:

– предупредительное снижение. Давление масла ниже уставки Рмпрi, где i - текущая ступень частоты вращения;

– аварийное снижение. Давление масла ниже уставки Рмаi.

Контроль давления масла включается через 10 с после запуска дизель-генератора.

Схема связей автомата и граф переходов приведены на рис. 41, 42.

4.3.10.2. *Схема связей*

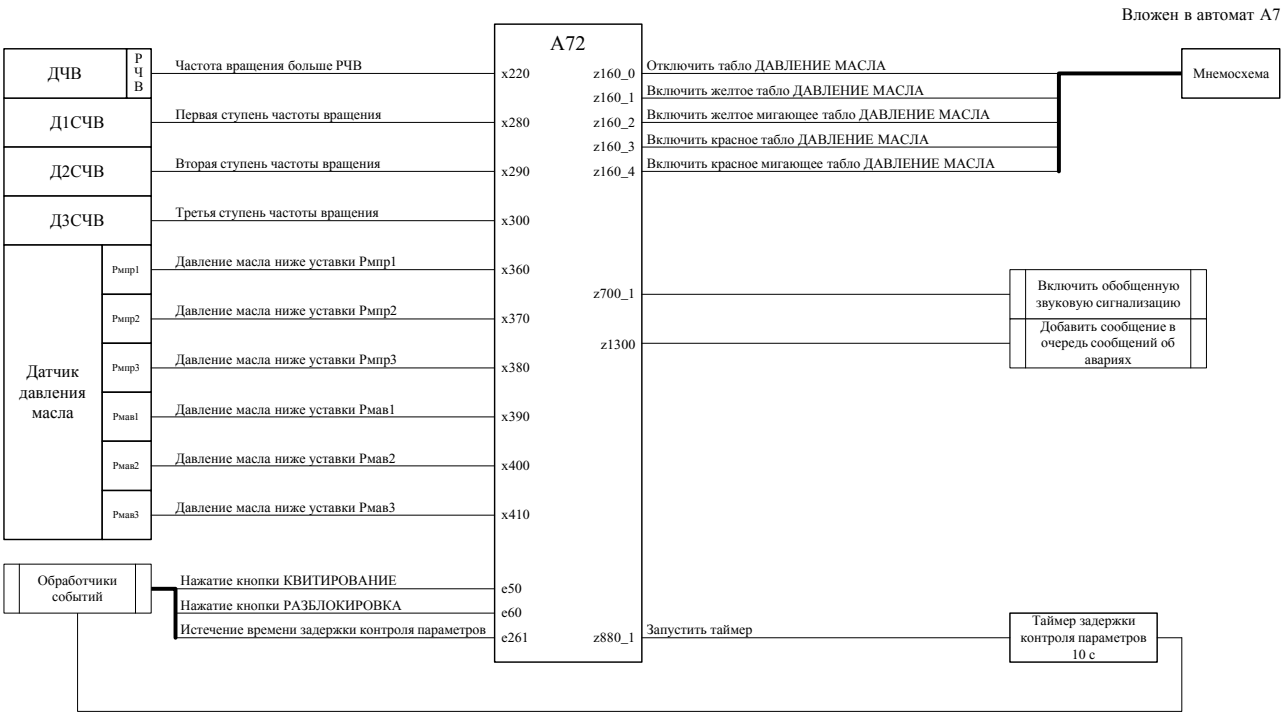

Рис. 41. Схема связей автомата контроля давления масла

4.3.10.3. *Граф переходов*

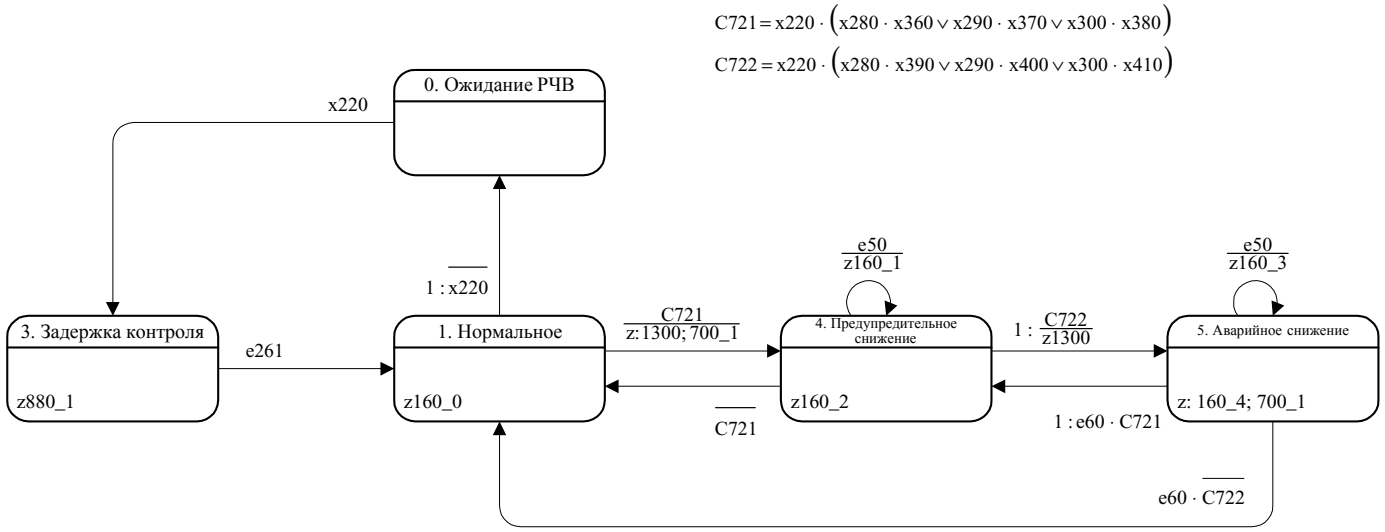

Рис. 42. Граф переходов автомата контроля давления масла

4.3.10.4. *Текст функции, реализующей автомат*

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

int C721( dg_t *dg )
{
  return   x220(dg) && (     ( x280(dg) && x360(dg) )
                          || ( x290(dg) && x370(dg) )
                          || ( x300(dg) && x380(dg) )
                        ) ;
} ;

int C722( dg_t *dg )
{
  return   x220(dg) && (     ( x280(dg) && x390(dg) )
                          || ( x290(dg) && x400(dg) )
                          || ( x300(dg) && x410(dg) )
                        ) ;
} ;

void A72( int e, dg_t *dg )
{
  int y_old = dg->y72 ;

  #ifdef GRAPH_EVENTS_LOGGING
    log_exec( dg, "A72", y_old, e ) ;
  #endif

  switch( dg->y72 )
  {
    case 0:
      if( x220(dg) )                          dg->y72 = 3 ;
    break ;

    case 1:
      if( !x220(dg) )                         dg->y72 = 0 ;
      else
      if( C721(dg) ) { z1300(dg, MSG_OIL_PRESS_ALARM) ; z700_1(dg) ;
                                              dg->y72 = 4 ; }
    break ;
```

```
    case 3:
      if( e == 261 )                          dg->y72 = 1 ;
    break ;

    case 4:
      if( C722(dg) ) { z1300(dg, MSG_OIL_PRESS_FAIL) ;
                                              dg->y72 = 5 ; }
      else
      if( !C721(dg) )                         dg->y72 = 1 ;
      else
      if( e == 50 ) { z160_1(dg) ; }
    break ;

    case 5:
      if( e == 60 && C721(dg) )               dg->y72 = 4 ;
      else
      if( e == 60 && !C722(dg) )              dg->y72 = 1 ;
      else
      if( e == 50 ) { z160_3(dg) ; }
    break ;

    default:
      #ifdef GRAPH_ERRORS_LOGGING
        log_write( LOG_GRAPH_ERROR, dg->number,
                   "ERROR IN A72: unknown state number!", 0 ) ;
      #endif
    break ;
  } ;

  if( y_old == dg->y72 ) goto A72_end ;

  {
    #ifdef GRAPH_TRANS_LOGGING
      log_trans( dg, "A72", y_old, dg->y72 ) ;
    #endif

    #ifdef DEBUG_FRAME
      update_debug() ;
    #endif
  } ;

  switch( dg->y72 )
  {
    case 1:
      z160_0(dg) ;
    break ;

    case 3:
      z880_1(dg) ;
    break ;

    case 4:
      z160_2(dg) ;
    break ;

    case 5:
      z160_4(dg) ; z700_1(dg) ;
    break ;
  } ;

  A72_end:
  #ifdef GRAPH_ENDS_LOGGING
    log_end( dg, "A72", dg->y72, e ) ;
  #endif
  ;
} ;
```

4.3.11. Протоколы функционирования системы управления

Приведем примеры протоколов работы для системы управления в целом при обработке нажатия кнопки

"Подготовка к пуску" (событие e10), полученный автоматически с выделенными вручную этапами.

Диагностирующий (полный) протокол имеет следующий вид.

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 маслопрокачивающий насос:

```
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 дизель-генератору:

```
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
```

Ниже приводится проверяющий (короткий) протокол обработки нажатия кнопки "Подготовка к пуску" (событие e10).

```
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
```

Для иллюстрации работы вложенных автоматов приведем структуру рассмотренного полного протокола.

```
1.  A20          e10 в 2          В состояние 2 автомата A20 вложены
                                  A7,A0,A5,A13,A62
2.  A7           e10 в 0          Вложен в состояние 2 автомата A20
3.  A71          e10 в 0          Вложен в состояние 0 автомата A7
    x320==0, x330==0
    A71          Завершил обработку e10 в состоянии 0
4.  A72          e10 в 0          Вложен в состояние 0 автомата A7
    x220==0
    A72          Завершил обработку e10 в состоянии 0
5.  A73          e10 в 0          Вложен в состояние 0 автомата A7
    x220==0
    A73          Завершил обработку e10 в состоянии 0
6.  A74          e10 в 0          Вложен в состояние 0 автомата A7
    x430==0, x440==0
    A74          Завершил обработку e10 в состоянии 0
7.  A75          e10 в 0          Вложен в состояние 0 автомата A7
    x460==0
    A75          Завершил обработку e10 в состоянии 0
8.  A76          e10 в 0          Вложен в состояние 0 автомата A7
    x470==0
    A76          Завершил обработку e10 в состоянии 0
9.  A77          e10 в 0          Вложен в состояние 0 автомата A7
    x480==0
    A77          Завершил обработку e10 в состоянии 0
10. A78          e10 в 0          Вложен в состояние 0 автомата A7
    x500==0
    A78          Завершил обработку e10 в состоянии 0
    x20==1, x260==0
    A7           Завершил обработку e10 в состоянии 0
```

```
11. A0          e10 в 0          Вложен в состояние 2 автомата A20
12. A8          e10 в 0          Вложен в состояние 0 автомата A0
    x220==0, x500==0, x510==0, x520==0, x700==0
    A8          0 → 2
    A8          Завершил обработку e10 в состоянии 2
    A0          0 → 2            В состояние 2 автомата A0 вложены
                                 A4, A3, A1
13. A4          e0 в 0           Вложен в состояние 2 автомата A0
14. A4_0        e0 в 0           Вложен в состояние 0 автомата A4
    A4_0        Завершил обработку e0 в состоянии 0
    A4          Завершил обработку e0 в состоянии 0
15. A3          e0 в 0           Вложен в состояние 2 автомата A0
    A3          Завершил обработку e0 в состоянии 0
16. A1          e0 в 0           Вложен в состояние 2 автомата A0
    A1          0 → 1            В состояние 1 автомата A1 вложен A1_0
17. A1_0        e0 в 0
    z800_0                       На петле в состоянии 0
    A1_0        Завершил обработку e0 в состоянии 0
    z10_1, z20_1, z30_3          В состояние 1 автомата A1
    z400_1                       В состояние 1 автомата A1
18. A80(z400)   e410 в 0
    x900(z400)==0
    A80(z400)   0 → 3
    z270_2(z400), z950_1(z400), z1000_1(z400)
    A80(z400)   Завершил обработку e410 в состоянии 3
    z420_1                       В состояние 1 автомата A1
    z430_1                       В состояние 1 автомата A1
19. A80(z430)   e410 в 0
    x900(z430)==0
    A80(z430)   0 → 3
    z270_2(z430), z950_1(z430), z1000_1(z430)
    A80(z430)   Завершил обработку e410 в состоянии 3
    z440_1                       В состояние 1 автомата A1
20. A80(z440)   e410 в 0
    x900(z440)==0
    A80(z440)   0 → 3
    z270_2(z440), z950_1(z440), z1000_1(z440)
    A80(z440)   Завершил обработку e410 в состоянии 3
    z460                         В состояние 1 автомата A1
21. A5          e380 в 0         Вызван из z460
    x810==1
    A5          0 → 2
    z610_1
    A5          Завершил обработку e380 в состоянии 2
    z810_1                       В состояние 1 автомата A1
    A1          Завершил обработку e0 в состоянии 1
    z290_0                       В состояние 2 автомата A0
    A0          Завершил обработку e0 в состоянии 2
22. A5          e10 в 2          Вложен в состояние 2 автомата A20
    x810==1, x820==0
    A5          Завершил обработку e10 в состоянии 2
23. A13         e10 в 0          Вложен в состояние 2 автомата A20
    A13         Завершил обработку e10 в состоянии 0
24. A62         e10 в 0          Вложен в состояние 2 автомата A20
    A62         0 → 1
    z1110_1, z1120_1, z1140
    A62         Завершил обработку e10 в состоянии 1
    A20         Завершил обработку e10 в состоянии 2
```

4.4. Программный имитатор дизель-генератора

Покажем, что предлагаемый подход применим не только для разработки программного обеспечения систем управления, но и для создания имитатора объекта управления, необходимого для автономной отладки системы управления.

Так как среди четырех разработанных автоматов между собой взаимодействуют только автоматы A30 и A31 (по вложенности), то схема взаимодействия автоматов не приводится.

Обратим внимание, что некоторые из реализующих имитатор автоматов осуществляют управление аналоговыми параметрами. Среди автоматов, рассмотренных ниже, автомат A20 выполняет имитацию изменения затяжки пружины регулятора частоты вращения, а автомат A31 имитирует изменение частоты вращения дизель-генератора. Это возможно благодаря тому, что входные переменные и выходные воздействия автоматов реализуются функциями, которые могут выполнять произвольные действия.

Также отметим, что в отличие от известных подходов, ориентированных на **моделирование** динамических систем с использованием гибридных автоматов (например, реализующих кусочно-непрерывные функции) [4], в рамках предлагаемой парадигмы при **программировании** применяются конечные автоматы с "расширенными" входами и выходами. При этом для описания поведения автоматов используется нотация, весьма близкая к классической, и применяются такие классические термины как, например, автоматы Мура и Мили.

4.4.1. Автомат имитации исполнительного устройства

4.4.1.1. *Словесное описание*

Автомат предназначен для имитации клапана (или любого аналогичного по логике работы исполнительного устройства, например, электродвигателя, используемого в рассматриваемой системе), снабженного сигнализаторами открытого и закрытого положения.

При построении автомата учитываются следующие особенности моделируемого устройства.

1. При поступлении команды открытия или закрытия клапана включается задержка времени исполнения команды, по истечении которой имитируется срабатывание сигнализатора соответствующего положения.

2. Если до истечения времени задержки исполнения команды поступает команда перевода клапана в исходное положение, задержка включается заново и выполняется отработка новой команды.

Схема связей автомата и граф переходов приведены на рис. 43, 44.

4.4.1.2. *Схема связей*

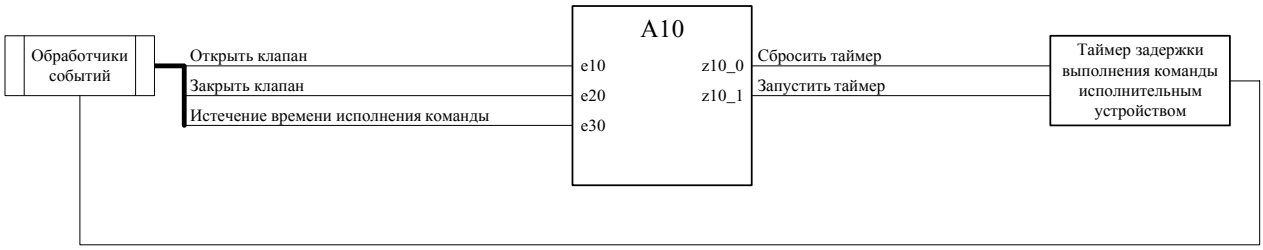

Рис. 43. Схема связей автомата имитации исполнительного устройства

4.4.1.3. *Граф переходов*

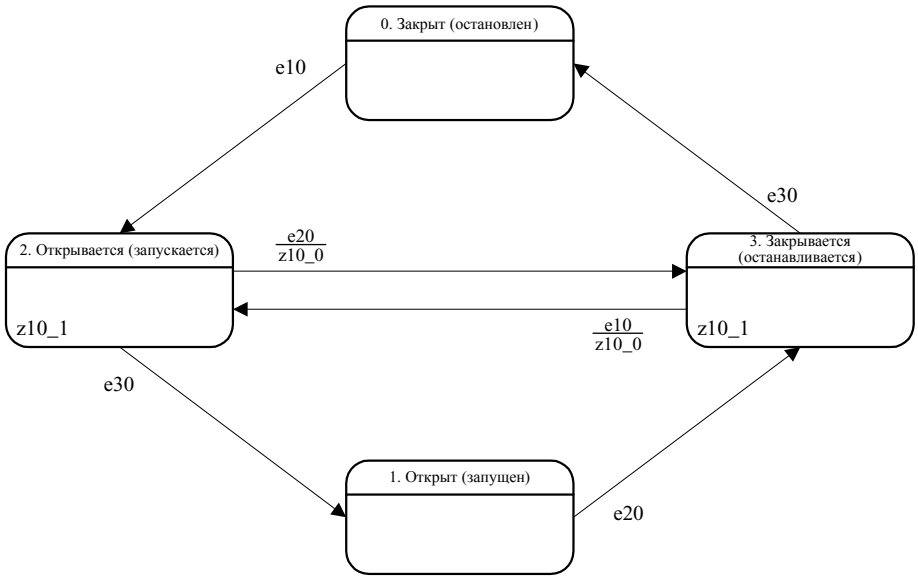

Рис. 44. Граф переходов автомата имитации исполнительного устройства

4.4.1.4. *Текст функции, реализующей автомат*

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

void A10( int e, dg_t *dg, valve_t *v )
{
  int y_old = v->y ;

  switch( v->y )
  {
    case 0:
      if( e == 10 )                 v->y = 2 ;
    break ;

    case 1:
      if( e == 20 )                 v->y = 3 ;
    break ;

    case 2:
      if( e == 30 )                 v->y = 1 ;
      else
      if( e == 20 ) { z10_0(dg,v) ; v->y = 3 ; } ;
    break ;

    case 3:
      if( e == 30 )                 v->y = 0 ;
      else
      if( e == 10 ) { z10_0(dg,v) ; v->y = 2 ; } ;
    break ;
  } ;

  if( y_old == v->y ) return ;

  switch( v->y )
  {
    case 2:
      z10_1(dg,v) ;
    break ;

    case 3:
      z10_1(dg,v) ;
    break ;
  } ;
} ;
```

4.4.2. Автомат имитации регулятора частоты вращения

4.4.2.1. *Словесное описание*

Автомат предназначен для имитации регулятора частоты вращения. Автомат обрабатывает команды на уменьшение или увеличение затяжки пружины регулятора, имитируя при этом срабатывание сигнализаторов крайних положений регулятора.

Схема связей автомата и граф переходов приведены на рис. 45, 46.

4.4.2.2. *Схема связей*

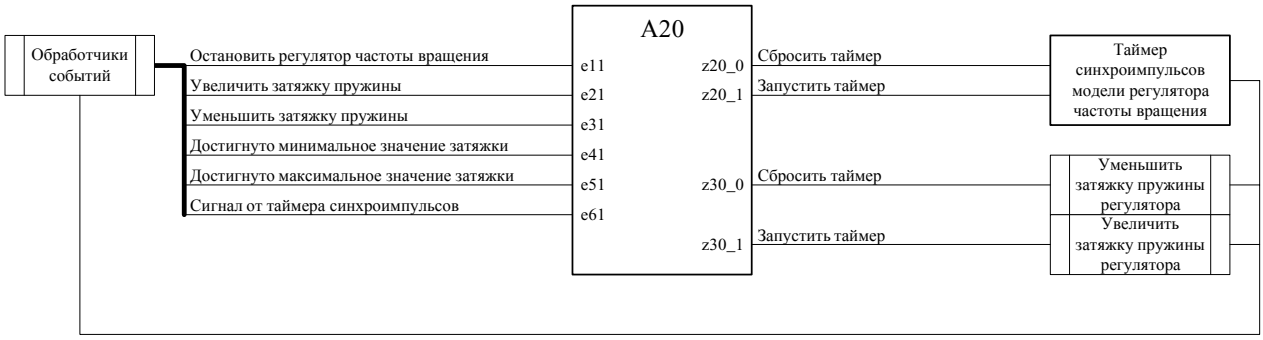

Рис. 45. Схема связей автомата имитации регулятора частоты вращения

4.4.2.3. *Граф переходов*

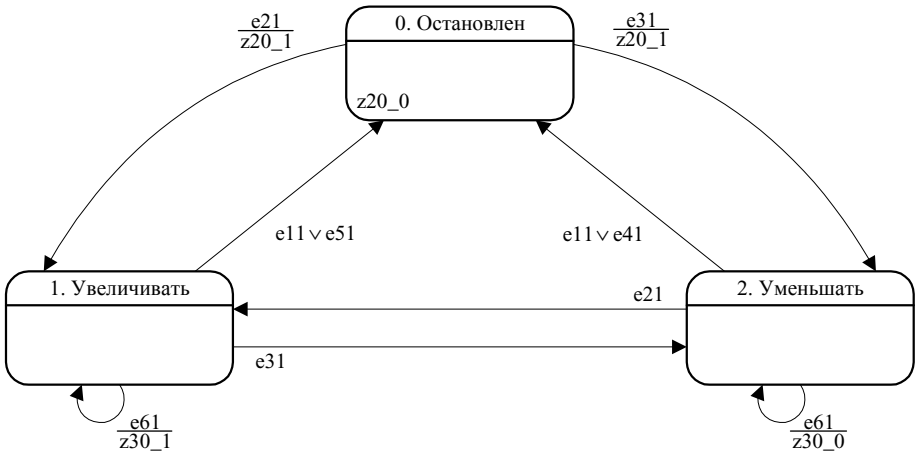

Рис. 46. Граф переходов автомата имитации регулятора частоты вращения

4.4.2.4. *Текст функции, реализующей автомат*

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

void A20( int e, dg_t *dg )
{
  int y_old = dg->y20 ;

  switch( dg->y20 )
  {
    case 0:
      if( e == 21 ) { z20_1(dg) ;    dg->y20 = 1 ; }
      else
      if( e == 31 ) { z20_1(dg) ;    dg->y20 = 2 ; }
    break ;

    case 1:
      if( e == 11 || e == 51 )       dg->y20 = 0 ;
      else
      if( e == 31 )                  dg->y20 = 2 ;
      else
      if( e == 61 ) { z30_1(dg) ; } ;
    break ;

    case 2:
      if( e == 11 || e == 41 )       dg->y20 = 0 ;
      else
      if( e == 21 )                  dg->y20 = 1 ;
      else
      if( e == 61 ) { z30_0(dg) ; } ;
    break ;
  } ;

  if( y_old == dg->y20 ) return ;

  switch( dg->y20 )
  {
    case 0:
      z20_0(dg) ;
    break ;
  } ;
} ;
```

4.4.3. Автомат имитации дизель-генератора

4.4.3.1. *Словесное описание*

Автомат предназначен для имитации основных состояний дизель-генератора. В нем выделены состояния, перечисленные ниже.

1. Остановлен.

2. Проворот. Имитатор дизель-генератора переходит в это состояние при срабатывании имитаторов сигнализаторов открытых положений клапанов подачи воздуха и подачи воздуха низкого давления. В этом состоянии с заданной

частотой имитируется выдача сигналов от датчика проворота коленвала.

3. Запуск. Имитатор дизель-генератора переходит в это состояние при срабатывании имитаторов сигнализаторов открытого положения клапана подачи воздуха и главного пускового клапана. При этом запускается задержка времени пуска, по истечении которой имитируется запуск дизель-генератора.

4. Работа. В этом состоянии выполняется алгоритм имитации изменения частоты вращения, описанный в следующем подразделе.

Схема связей автомата и граф переходов приведены на рис. 47, 48.

4.4.3.2. *Схема связей*

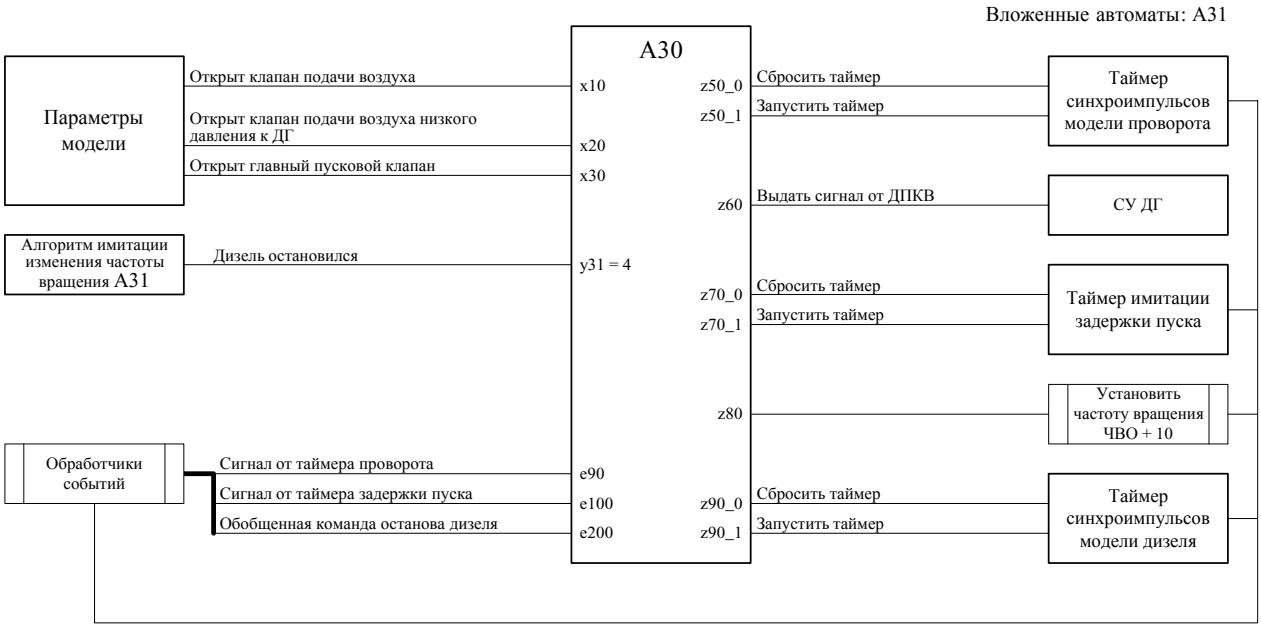

Рис. 47. Схема связей автомата имитации дизель-генератора

4.4.3.3. *Граф переходов*

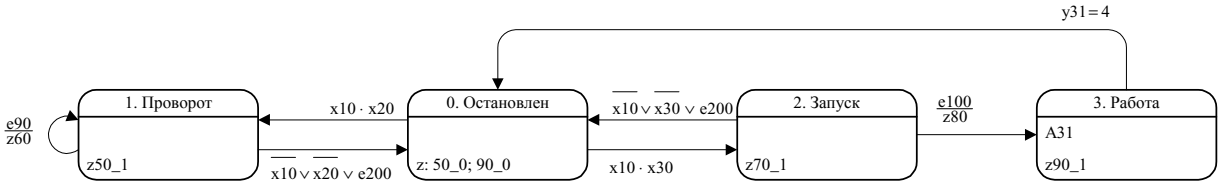

Рис. 48. Граф переходов автомата имитации дизель-генератора

4.4.3.4. *Текст функции, реализующей автомат*

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

void A30( int e, dg_t *dg )
{
  int y_old = dg->y30 ;

  switch( dg->y30 )
  {
    case 0:
      if( x10(dg) && x20(dg) )                dg->y30 = 1 ;
      else
      if( x10(dg) && x30(dg) )                dg->y30 = 2 ;
    break ;

    case 1:
      if( !x10(dg) || !x20(dg) || e == 200 )  dg->y30 = 0 ;
      else
      if( e == 90 ) { z60(dg) ; } ;
    break ;

    case 2:
      if( !x10(dg) || !x30(dg) || e == 200 )  dg->y30 = 0 ;
      else
      if( e == 100 ) { z80(dg) ;              dg->y30 = 3 ; } ;
    break ;

    case 3:
      A31( e, dg ) ;
      if( dg->y31 == 4 )                      dg->y30 = 0 ;
    break ;
  } ;

  if( y_old == dg->y30 ) return ;

  switch( dg->y30 )
  {
    case 0:
      z50_0(dg) ; z90_0(dg) ;
    break ;

    case 1:
      z50_1(dg) ;
    break ;

    case 2:
      z70_1(dg) ;
    break ;

    case 3:
      A31(0,dg) ;
      z90_1(dg) ;
    break ;
  } ;
} ;
```

4.4.4. Автомат имитации изменения частоты вращения

4.4.4.1. *Словесное описание*

Автомат предназначен для имитации изменения частоты вращения дизель-генератора.

Если текущая частота вращения дизель-генератора отличается от установленной на регуляторе, то имитируется плавное изменение частоты вращения.

При поступлении от системы управления команды на останов дизель-генератора (этой командой считается команда на закрытие наружной захлопки газоотвода), выполняется быстрое уменьшение его частоты вращения до нуля.

Схема связей автомата и граф переходов приведены на рис. 49, 50.

4.4.4.2. *Схема связей*

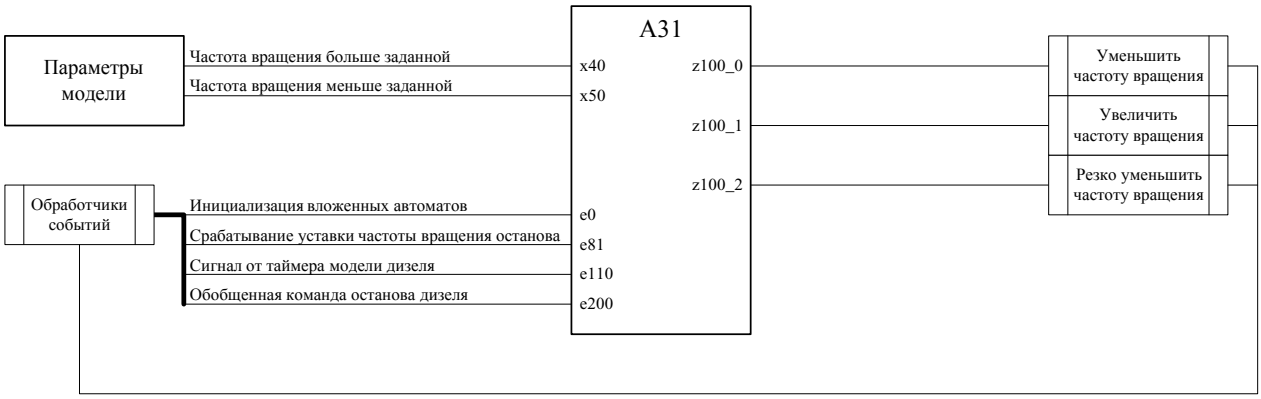

Рис. 49. Схема связей автомата имитации изменения частоты вращения

4.4.4.3. *Граф переходов*

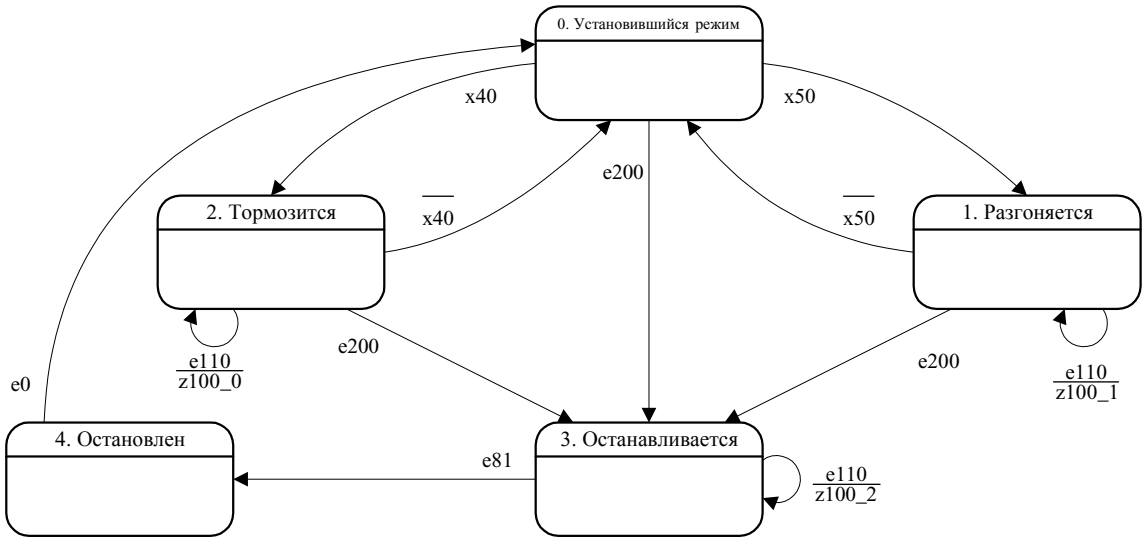

Рис. 50. Граф переходов автомата имитации изменения частоты вращения

4.4.4.4. *Текст функции, реализующей автомат*

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

void A31( int e, dg_t *dg )
{
  int y_old = dg->y31 ;

  switch( dg->y31 )
  {
    case 0:
      if( x40(dg) )           dg->y31 = 2 ;
      else
      if( x50(dg) )           dg->y31 = 1 ;
      else
      if( e == 200 )          dg->y31 = 3 ;
    break ;

    case 1:
      if( !x50(dg) )          dg->y31 = 0 ;
      else
      if( e == 110 ) { z100_1(dg) ; }
      else
      if( e == 200 )          dg->y31 = 3 ;
    break ;

    case 2:
      if( !x40(dg) )          dg->y31 = 0 ;
      else
      if( e == 110 ) { z100_0(dg) ; }
      else
      if( e == 200 )          dg->y31 = 3 ;
    break ;

    case 3:
      if( e == 81 )           dg->y31 = 4 ;
      else
      if( e == 110 ) { z100_2(dg) ; } ;
    break ;

    case 4:
      if( e == 0 )            dg->y31 = 0 ;
    break ;
  } ;
} ;
```

5. Заключение

В результате исследований по теме "Разработка основных положений технологии создания программного обеспечения "реактивных" систем", выполненных по второму этапу темы "Разработка технологии создания программного обеспечения систем управления на основе автоматного подхода", сформулированы основные положения технологии, поддерживающей для рассматриваемого класса систем все этапы создания программного обеспечения, к которым относятся: изучение предметной области, анализ, проектирование, реализация, отладка, сертификация и документирование.

К таким положениям могут быть отнесены следующие:

– четко выделен этап проектирования, предшествующий этапу реализации. При этом строятся схемы и диаграммы, которые в дальнейшем формально и изоморфно преобразуются в текст программы. Такая организация процесса разработки аналогична принятой при создании аппаратуры, изготовление которой выполняется только по проектной документации;

– опыт использования предлагаемой технологии показывает, что при тщательном проектировании построенная на этапе реализации программа либо сразу работает правильно, либо требует существенно меньшего времени на отладку, чем при традиционном подходе;

– расширено по отношению к системам логического управления понятие "входное воздействие" за счет использования "событий", которые в отличие от "входных переменных", не опрашиваются программой, а вызывают соответствующие им обработчики;

– расширены понятия "входная переменная" и "выходное воздействие" за счет перехода от двоичных переменных к произвольным подпрограммам (функциям);

– разработана новая структурная схема событийных систем, в которой повышен уровень централизации управления за счет выноса логики из обработчиков событий с целью формирования системы взаимосвязанных автоматов, которые при необходимости вызываются из обработчиков событий с передачей автоматам соответствующих событий;

– дополнена нотация, применяемая при построении графов переходов, например, за счет перечисления вложенных автоматов;

– в отличие от объектно-ориентированного проектирования построение всех основных моделей основано на применении только автоматной терминологии, а для описания динамики используется модель только одного типа — система взаимосвязанных графов переходов. При этом автоматы в явном виде используются на разных этапах создания программного обеспечения: проектировании, программировании, отладки и документировании;

– применение автоматов в предлагаемой нотации в качестве динамической модели позволяет эффективно описывать и реализовывать задачи рассматриваемого класса даже при большой их размерности. Использование графов переходов в качестве языка спецификации 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 графа переходов и изоморфной ему подпрограммы обеспечивает понимание последней;

– разработан универсальный алгоритм программной реализации иерархии графов переходов для произвольного их количества и произвольного уровня вложенности;

– каждый граф переходов формально и изоморфно реализуется по шаблону в виде подпрограммы на выбранном языке программирования. Указанный изоморфизм позволяет решить обратную задачу — однозначно восстановить граф переходов по этой подпрограмме;

– автоматическое ведение протокола в терминах спецификации (автоматов) обеспечивает возможность сертификации программы. При этом для рассматриваемых событий при выбранных значениях входных переменных демонстрируется соответствие функционирования программы "поведению", определяемому выходными воздействиями и состояниями автоматов, которые отражаются в "полном" протоколе. Совокупность "полных" протоколов обеспечивает возможность сертификации программы в целом. Для сертификации в терминах, понятных Заказчику, могут применяться "короткие" протоколы, в которых указываются, например, только формируемые выходные воздействия;

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

– порождаемый некоторыми событиями протокол или его часть является соответствующим сценарием. Таким образом, в рамках разработанной технологии сценарий строится автоматически по построенной программе, а не вручную при ее синтезе, как это предлагается делать, например, при использовании языка UML. Это объясняется тем, что ручное построение всей совокупности сценариев и формальный

синтез системонезависимой части программы по ним для задач со сложным поведением практически невозможно;

– кроме отладки и сертификации программы с использованием протоколов, возможен также традиционный интерактивный подход и отладка на основе введенного в [1] свойства наблюдаемости программ (по номерам состояний автоматов);

– без использования объектного подхода программа четко разделяется на две части — системонезависимую и системозависимую;

– этапы проектирования и реализации системонезависимой части программы полностью разделены;

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

– реализация входных переменных и выходных воздействий в виде функций обеспечивает: абстрагирование, декомпозицию, протоколирование, упрощение внесения изменений, простоту перехода от одних типов источников и приемников информации к другим, а также наличие действующего макета программы в любой момент времени после начала реализации системозависимой части;

– подробное документирование проекта создания программного обеспечения упрощает внесение изменений в него даже через длительный срок после выпуска, в том числе и специалистами, не участвовавшими в проектировании. При этом изменения должны вноситься в весь комплект документации, а не только в его программную часть. Внесение изменений в системонезависимую часть должно начинаться с корректировки графов переходов, структура которых весьма удобна для внесения изменений. Внесенные в графы переходов изменения должны быть

формально и изоморфно отражены в соответствующих программных модулях.

В заключение отметим, что в предыдущие десятилетия большинство программистов имели либо инженерное, либо математическое образование с соответствующей, устоявшейся веками, культурой. Современные программисты воспитываются иначе [12], а дисциплине программирования должного внимания не уделяется.

Предлагаемая технология является новой попыткой введения такой дисциплины и основана на априорном задании требуемых состояний и их визуализации.

Опыт применения предлагаемой технологии подтвердил следующее высказывание: "то, что не специфицировано формально, не может быть проверено, а то, что не может быть проверено, не может быть безошибочным" [13]. Поэтому авторы надеются (особенно учитывая мнение о работе [1], высказанное в [14]), что предложенный подход по крайней мере для систем логического управления и "реактивных" систем является приближением к "серебряной пуле" [8] в части создания качественных программ, тем более, что Ф. Брукс отозвался благосклонно только о подходе Д. Харела [5], также основанном на применении автоматов, преимущество по сравнению с которым показано в [11].

Целесообразность использования автоматного подхода подтверждается 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" [15].

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

Авторы считают, что предложенный подход для рассматриваемых классов систем, весьма распространенных на практике, в соответствии с принципом Оккама "не размножает сущности без необходимости" (как, например, UML) и обладает "минимализмом" [16], достаточным для обеспечения понимаемости программ специалистами, не являющимися их разработчиками.

Отметим также, что предложенная в настоящей работе парадигма существенно отличается от изложенной в [4]. Основное отличие заключается в том, что нами предлагается технология, не зависящая от сред разработки и реализации, а в [4] рассмотрены программные продукты для моделирования.

Рассматриваемая в настоящей работе парадигма автоматного программирования, которая может быть названа также "программирование с явным выделением состояний", начинает все шире использоваться [17—19]. Для большей ее популяризации на сайте [20] создан раздел "Автоматы".

6. Публикации по результатам этапа

По результатам выполненных в ходе этапа работ опубликованы следующие статьи:

1. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем //Промышленные АСУ и контроллеры. 2000. №10. С. 44-48.

2. Шалыто А.А., Туккель Н.И. Автоматный подход к созданию программного обеспечения для систем логического управления и "реактивных" систем //Системы управления и обработки. ФГУП "НПО "Аврора"". 2000. Вып. 2. С. 165-173.
3. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и "реактивных" систем (обзор) //Автоматика и телемеханика. 2001. №1. С. 3-39.
4. Шалыто А.А., Туккель Н.И. Программирование с явным выделением состояний //Мир ПК. 2001. №8. С. 116-121, №9. С. 132-138.
5. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем //Известия ВУЗов. Приборостроение. 2001. №9. С. 28-35.
6. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. №5. С. 45-62.

По результатам выполненных в ходе этапа работ в материалах конференций опубликованы:

1. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем /Тезисы докладов Международной научно-методической конференции "Телематика-2000". СПб.: СПбГИТМО (ТУ), 2000. С. 88-91.
2. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем /Материалы международной научно-технической конференции "Кибернетика и технологии XXI века". Воронеж: ВГТУ, 2000. С. 308-316.

3. Шалыто А.А., Туккель Н.И., Ваганов С.А. Повышение централизации управления при программировании "реактивных" систем /Труды международной научно-методической конференции "Телематика-2001". СПб.: СПбГИТО (ТУ), 2001. С. 174-176.
4. Туккель Н.И., Шалыто А.А. Применение автоматного программирования для создания "реактивных" систем управления /Материалы международной научно-практической конференции "Математическое моделирование в образовании, науке и производстве". Тирасполь: Приднепровский гос. университет им. Т.Г.Шевченко, 2001. С. 469-471.

Кроме того, по результатам выполненных в ходе этапа работ были сделаны доклады на конференциях, материалы которых в настоящее время не опубликованы:

1. Шалыто А.А., Туккель Н.И. Применение автоматного проектирования программ в событийных системах /4-я международная конференция по морским интеллектуальным технологиям "Моринтех-2001". СПб.: НИЦ "Морские Интеллектуальные Технологии", 2001.
2. Туккель Н.И., Шалыто А.А. Применение автоматного проектирования программ в "реактивных" системах /Третья международная конференция по проблеме "Логико-лингвистического управления динамическими объектами. СПб.: Институт проблем машиноведения РАН, 2001.

Список литературы

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
2. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.:Наука, 2000. 773 с.
3. Шалыто А.А., Туккель Н.И. SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5. С. 29-35.
4. Бенькович Е.С., Колесов Ю.Б., Сениченков Ю.Б. Практическое моделирование динамических систем. СПб.: БХВ-Петербург, 2002. 464 с.
5. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК, 2000. 432 с.
6. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 366 с.
7. 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.
8. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. СПб.: Символ, 2000. 304 с.
9. Ершов А.П. Смешанные вычисления //В мире науки. 1984. ¹ 6.
10. Лавров С.С. Программирование. Математические основы, средства, теория. СПб.: БХВ-Петербург, 2001. 320 с.
11. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и "реактивных" систем // Автоматика и телемеханика. 2001. №1. С.3-35.
12. Черняк Л. Создание программ как инженерная дисциплина //COMPUTERWORLD РОССИЯ. 2000. №37. С.18-20.
13. Зайцев С.С. Описание и реализация протоколов сетей ЭВМ. М.: Наука, 1989. 112 с.
14. Герр Р. Новый поворот // PC Magazine / Russian Edition. 1998. №10. С.88-90.
15. Кук Д., Урбан Д., Хамилтон С. Unix и не только. Интервью с Кеном Томпсоном //Открытые системы. 1999. №4. С.35-47.
16. Герр Р. Отладка человечества // PC Magazine / Russian Edition. 2000. №5. С.90-91.
17. Богатырев Р. Об асинхронном и автоматном программировании //Открытые системы. 2001. N3. С.68-69.
18. Любченко В.С. Мы выбираем, нас выбирают... (к проблеме выбора алгоритмической модели) //Мир ПК. 1999. N3. С.38-40.
19. Кузнецов Б.П. Психология автоматного программирования //BYTE/Россия. 2000. N11. С.22-29.
20. http://www.softcraft.ru

Глоссарий

Автомат	Совокупность множества состояний и множества переходов между состояниями. Условия переходов формируются из множества входных воздействий. В состояниях, на переходах и петлях могут формироваться выходные воздействия
Автомат Мили	Автомат, формирующий выходные воздействия на переходах
Автомат Мура	Автомат, формирующий выходные воздействия в состояниях
Активация вложенного автомата	В общем случае выполняется при переходе головного автомата в состояние, в которое вложен рассматриваемый автомат. Активация заключается в запуске вложенного автомата со специальным внутренним событием e0. В основном используется для перевода автомата в начальное состояние
Бесповторность опроса входных переменных	Требование, заключающееся в том, что при проверке условий перехода из состояния автомата, каждая входная переменная должна опрашиваться не более одного раза (проблема "риска").
Вершина графа переходов	Прямоугольник с закругленными углами (или окружность), обозначающий состояние автомата. Вершина может помечаться номером состояния и его названием, перечнем формируемых выходных воздействий и перечнем вложенных автоматов
Вложенный автомат	Автомат, запускаемый из одного или нескольких состояний головного автомата при запуске последнего. Вложенный автомат запускается с тем же событием, что и головной
Внутреннее событие	Событие, порождаемое внутри системы взаимосвязанных автоматов
Входная переменная	Входное воздействие, опрашиваемое автоматом
Входное воздействие	Воздействие, при появлении которого автомат может изменить состояние. Входные воздействия делятся на события и входные переменные. Некоторые входные воздействия могут быть одновременно и событием и входной переменной. Разновидностью входных переменных являются предикаты, проверяющие номера состояний других автоматов
Входящая дуга	Дуга, входящая в рассматриваемую вершину
Вызываемый автомат	Автомат, запускаемый из выходного воздействия другого автомата в общем случае с передачей внутреннего события
Выходное воздействие	Воздействие, подаваемое автоматом на объекты управления
Генерирующий контур	Контур в графе переходов, условия на дугах которого могут быть одновременно истинны
Головной автомат	Автомат является головным для автомата, который в него вложен
Граф переходов	Диаграмма, задающая поведение автомата в терминах входных воздействий, состояний, переходов и выходных воздействий
Групповая дуга	Дуга, заменяющая для группы вершин дуги с одинаковыми условиями переходов, выходными воздействиями и конечными вершинами

Групповая петля	Петля, заменяющая для группы вершин петли с одинаковыми условиями переходов и выходными воздействиями
Диагностирующий (полный) протокол	Протокол, содержащий записи о запуске автоматов, завершении их работы, опросе входных воздействий, формировании выходных воздействий, изменении состояний автоматов. Этот протокол может быть использован для локализации ошибок
Достижимость	Свойство графа переходов, заключающееся в существовании хотя бы одного пути из каждой вершины в любую другую
Дуга графа переходов	Линия, заканчивающаяся стрелкой, которая связывает две вершины в графе переходов. Дуга может помечаться приоритетом, условием перехода и перечнем формируемых выходных воздействий
Завершение работы автомата	Завершение работы подпрограммы, реализующей автомат
Запуск автомата	Вызов подпрограммы, реализующей автомат, с передачей в качестве параметра номера произошедшего события и, если необходимо, информации о нем
Исходящая дуга	Дуга, исходящая из рассматриваемой вершины
Кодирование состояний	Присвоение состояниям уникальных (численных) идентификаторов, позволяющих различать эти состояния. Этап предлагаемой технологии
Многозначное кодирование состояний	Разновидность кодирования состояний, при использовании которого переменной состояния автомата присваивается десятичное значение
Наблюдаемость программы	Возможность однозначно определить состояние программы в любой момент времени путем слежения за номерами состояний системы взаимодействующих автоматов
Непротиворечивость	Свойство графа переходов, заключающееся в том, что условия переходов на исходящих из вершины дугах и петлях не могут быть истинными одновременно. Это может быть обеспечено расстановкой приоритетов или ортогонализацией, которая может быть физической или математической
Обмен номерами состояний	Способ взаимодействия автоматов, при котором один автомат опрашивает значение переменной состояния другого автомата
Обработчик события	Подпрограмма, вызываемая при возникновении определенного события. Обработчики событий запускают головной автомат системы взаимосвязанных автоматов
Оператор `switch`	Конструкция языка Си (или аналогичная конструкция другого языка), используемая для изоморфной реализации автомата, определившая название технологии. В операторе `switch` используется переменная состояния, а каждая метка `case` соответствует номеру состояния автомата
Опрос входных переменных	Обращение из программы к источникам информации, рассматриваемым как входные переменные, с целью определения их текущего значения
Переменная состояния автомата	Переменная, позволяющая различать состояния автомата. Данная переменная используется в операторе `switch`
Переход в автомате	Изменение состояния автомата
Петля графа переходов	Частный случай дуги графа переходов с одной и той же начальной и конечной вершиной

Полнота	Свойство графа переходов, заключающееся в том, что конъюнкция условий переходов на исходящих из вершины дугах и петлях равна единице
Приоритет перехода	Числовое обозначение на графе, определяющее порядок проверки условий переходов на дугах, исходящих из рассматриваемой вершины
Проблема "риска"	Если при проверке условий на исходящих из вершины дугах одна и та же входная переменная опрашивается более одного раза, то существует возможность, что переменная изменит свое значение между этими опросами. Такая ситуации названа проблемой "риска" (бесповторность опроса входных переменных).
Проверяющий (короткий) протокол	Протокол, содержащий записи о запуске головного автомата, завершении его работы и перечисление выходных воздействий, формируемых системой взаимосвязанных автоматов при обработке событий
Протокол	История реакции системы взаимодействующих автоматов на события с указанием времени занесения каждой записи. Протокол ведется в терминах автоматов и может содержать записи о запуске автоматов, завершении их работы, опросе входных воздействий, формировании выходных воздействий, изменении состояний автоматов
Реализация автомата	Создаваемая по шаблону подпрограмма, реализующая автомат
Системозависимая часть программы	Часть программы, написанная на уровне взаимодействия с операционной системой, инструментальной средой, аппаратурой и т.д.
Системонезависимая часть программы	Часть программы, не зависящая от операционной системы, инструментальной среды, аппаратуры и т.д.
Словесное описание автомата	Текстовый документ, содержащий не формализованное описание автомата. Служит для получения общих сведений о рассматриваемом автомате
Смешанный автомат	Автомат, формирующий выходные воздействия на переходах и в состояниях
Событие	Кратковременное входное воздействие, запускающее автомат
Событийная ("реактивная") система	Система, реагирующая на события
Состояние	Абстракция, соответсвующая стадии выполнения алгоритма
Состояние программы	Совокупность значений всех переменных, присутствующих в программе. Для программы, разработанной с использованием предлагаемой технологии — совокупность значений переменных состояния всех автоматов, которое, в свою очередь, может быть представлено одним десятичным числов
Схема взаимодействия автоматов	Диаграмма, формально описывающая взаимодействие автоматов по вложенности, вызываемости и обмену номерами состояний
Схема связей автомата	Диаграмма, формально описывающая входные и выходные воздействия автомата, содержащая также указание в какой автомат вложен рассматриваемый и какие автоматы вложены в него