С авторами можно связаться по адресу: Shalyto@mail.ifmo.ru

Сокращенная версия статьи опубликована с названием "Танки и автоматы" в журнале "BYTE/Россия", 2003. №2. С.69–73.

А В Т О М А Т Ы И Т А Н К И

Объектно-ориентированное программирование с явным выделением состояний

Анатолий Шалыто, Никита Туккель

Предложена методика совместного применения объектно-ориентированной и автоматной парадигм программирования, позволяющая эффективно описывать как структуру, так и поведение программ. Выполнено сравнение предлагаемого процесса проектирования с принятым при использовании языка *UML*. Подход иллюстрируется примером программирования танка для игры "Robocode".

При создании объектно-ориентированных программ [1,2] в литературе все большее внимание уделяется их проектированию [3–5]. Однако на практике с проектированием программы обычно "мучаются не долго, и поэтому мучаются всю жизнь" на остальных этапах жизненного цикла программы. Это во многом связано с недостаточной эффективностью известных технологий проектирования. В настоящей работе делается попытка разработки технологии проектирования программ рассматриваемого класса на основе минимального набора документов, описывающих как структурные, так и поведенческие стороны программ. При этом совместно применяются объектно-ориентированный и автоматный [6] подходы, образующие технологию, которая может быть названа "**объектно-ориентированное программирование с явным выделением состояний**". Эта технология иллюстрируется примером разработки системы управления виртуальным танком в игре "Robocode".

АВТОМАТЫ

При объектном проектировании для описания поведения объектов наряду с другими моделями [7,8] используется модель конечного детерминированного автомата, которую в дальнейшем будем называть "автоматом".

Эта модель, в отличие от применяемой в трансляторах [9], содержит выходы, на каждом из которых реализуется функция (подпрограмма, выполняющая определенные действия) [6,10]. Это расширяет класс задач [11], в котором могут быть применены автоматы. Более того, в работе [12] для описания поведения управляющих "устройств" было предложено использовать не одиночные автоматы, а системы взаимосвязанных автоматов, которые обладают широкими функциональными возможностями [13], например, в части реализации параллельных процессов. Автоматы в системе могут взаимодействовать на основе вложенности, вызываемости или обмена номерами состояний.

В автоматах в качестве входных воздействий могут использоваться:

– переменные, являющиеся функциями, возвращаемые значения которых являются значениями этих переменных;

– предикаты, проверяющие номера состояний автоматов, взаимодействующих с рассматриваемым автоматом;

– события, обработчики которых вызывают функции, реализующие автоматы, передавая им номера этих событий, что кратко можно сформулировано как "события, с которыми запускаются автоматы" [6,10]. При этом отметим, что в общем случае события могут быть как внешними, так и внутренними.

Изложенное опровергает сложившееся мнение о том, что автоматы имеют ограниченную область применения, например такую, как распознавание регулярных языков [9]. В изложенной выше трактовке область использования автоматов простирается на широкий класс задач, связанных, по крайней мере, с управлением, понимаемым весьма широко. К такому классу принадлежит и задача, рассматриваемая в настоящей работе.

На практике автоматы обычно задаются визуально в виде графов переходов (диаграмм состояний), причем существуют различные их модификации, например, предложенная в работе [14] и применяемая в работе [8]. В настоящей работе используются графы переходов смешанных автоматов (С-автоматов), нотация для построения которых предложена в работах [10,11].

В работах [3–8] вопрос о реализации автоматов в рамках применяемых методологий в должной мере не рассматривался. В работах [15,16] приведены примеры реализации автоматов в рамках объектной парадигмы. В первом случае используется компилятивный подход, на основе которого для каждого события записывается оператор switch, реализующий инициируемые этим событием переходы, что, по нашему мнению, делать нецелесообразно, так как реализация автомата оказывается распределенной по обработчикам событий. Во втором случае используется интерпретационный подход, применение которого ограничено избыточностью, присущей интерпретации [17].

В работе [18] было предложено, используя компилятивный подход, выполнять реализацию не для отдельных событий, а для графов переходов в целом. При этом граф переходов предлагается реализовывать не только формально, но и изоморфно, что достигается применением одного или двух операторов switch или аналогов этого оператора [12]. Указанный подход обеспечивает изобразительную эквивалентность спецификации и реализации, что позволяет начинать проверку этой части программы со сравнения ее текста с графом переходов.

Для событийных ("реактивных") систем авторами в работах [6,10] был предложен образец (шаблон), состоящий из двух операторов switch, предназначенный для формальной реализации графов переходов смешанных автоматов, которые строятся на основе указанной выше нотации. В этой связи следует обратить внимание на развиваемый в настоящее время в рамках объектной парадигмы подход — проектирование по образцам. Так, в частности, в работе [19] описан паттерн "State", обеспечивающий построение программ, в которых логика распределена по объектам, соответствующим состояниям. Нами развивается в некотором смысле противоположный подход [20], который направлен на повышение централизации логики в программах. Это связано с тем, что при "сильно" распределенном управлении понять поведение программы по ее тексту весьма трудно.

На базе изложенного, используя процедурный ("неклассовый") подход, в работе [12] был предложен вариант технологии создания программного обеспечения для систем логического управления. Эта технология была названа "SWITCH-технология" [21]. В дальнейшем был создан ее второй вариант, который предназначен для построения событийных систем [10].
Этот подход был назван в работе [6] "программирование с явным выделением состояний".

Он охватывает все этапы создания программного обеспечения, включая сертификацию и документирование.

В отличие от сред разработки, использующих автоматы при визуальном программировании (например, пакет "Stateflow" [22]), указанный подход является более универсальным, так как не зависит от применяемых инструментальных средств.

В настоящей работе предлагается еще один вариант автоматной технологии, предназначенный для объектно-ориентированного программирования (проектирования), который может быть назван "объектно-ориентированное программирование с **явным** выделением состояний".

Следует отметить, что несмотря на использование автоматов в указанных выше и в некоторых других методологиях объектной разработки программ [23], явное выделение состояний не характерно для объектной парадигмы. Например, работе [24], одним из критериев объектно-ориентированного языка является "поддержание им объектов как абстракции данных с определенным интерфейсом поименованных операций и **скрытым** состоянием". Да и само определение объекта, предложенное одним из "трех друзей" (являющихся создателями объектно-ориентированного проектирования) А. Джекобсоном, в этом смысле мало что проясняет: "Объект — это сущность, способная сохранять свое состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния" [25]. Кроме того, Г. Буч (один из "пророков-коммерсантов" [26]) пишет: "... поведение объекта определяется его историей: важна последовательность совершаемых над объектом действий. Это объясняется тем, что у объекта есть внутреннее состояние, которое характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущим (обычно динамическими) значениями каждого из этих свойств" [4].

Определение поведения как набора операций (методов) является традиционным в объектно-ориентированном программировании [23]. Оно сдерживает применение автоматов в рамках этой парадигмы программирования, несмотря на приведенное выше мнение Г. Буча о поведении объекта как **последовательности** действий. Это определение не позволяет конструктивно использовать автоматы, так как при большом числе свойств (атрибутов) количество состояний объекта может быть огромным, что делает невозможным выделение состояний в явном виде. Не лучше складывается ситуация с использованием автоматов и при определении состояния программы как совокупности значений ее всех переменных [27].

Предложения авторов по выходу из этой, казалось бы, тупиковой ситуации будут приведены ниже.

В заключение раздела отметим, что различие между алгоритмической и объектной декомпозициями имеет место, но оно не столь принципиально, как утверждает Г. Буч в работах [3,4]. В работах [12,28] предложено такое построение схем алгоритмов (за счет введения переменных состояния), что они будут изоморфны конструкции switch и графам переходов автоматов. Автоматы, в соответствии с приведенным выше определением объекта, сами могут рассматриваться как объекты, либо использоваться при описании их поведения.

ОСОБЕННОСТИ ТЕХНОЛОГИИ

Указанные в предыдущем разделе проблемы снимаются, если:

– атрибуты объекта разделить на управляющие и остальные по аналогии с организацией машины Тьюринга или машины фон Неймана [11];

– явно выделять состояния в управляющей части объекта, так как число "управляющих" (автоматных) состояний обычно значительно меньше числа остальных состояний, например,

"вычислительных". При этом находясь в одном из "управляющих" состояний, объект может пройти множество "вычислительных" состояний;

– ввести в программирование понятие "пространство состояний", понимая под ним множество состояний, описывающих поведение по крайней мере одного автомата. Ориентация в этом пространстве обеспечивает существенно более понятное поведение по сравнению со случаем, когда такое пространство или ориентация в нем отсутствует;

– поведение определять не только как реакцию на входные воздействия, состоящую в формировании выходных воздействий, но и через состояния и переходы автоматов, в которых эти входные и выходные воздействия применяются. Состояние является более важной и информативной абстракцией, определяющей поведение, по сравнению с входо-выходными последовательностями, так как в большинстве случаев для предсказания реакции достаточно знать только название состояния;

– "привязать" выходные воздействия к переходам, петлям или состояниям автомата;

– при спецификации задачи рассматривать автоматные состояния в качестве абстракций, переходя к переменным, им соответствующим, только на этапе реализации. Это позволяет программисту перестать быть "фокусником", отказавшись от неупорядоченного введения "флагов" при программировании. Программы с флагами, а не с явно выделенными состояниями, также "неустойчивы", как и слоны на тонких ножках, изображенные на картинах С. Дали, по сравнению с нормальными слонами. Однако такие слоны, в отличие от программ с флагами, не встречаются повсеместно;

– ввести в этап реализации программ, называемый традиционно "кодирование", **новый подэтап**, называемый в теории автоматов "кодирование состояний";

– использовать многозначное кодирование для каждого автомата с целью различия его состояний по значениям одной переменной;

– ввести в программирование понятие "наблюдаемость", обеспечивая возможность слежения за переходами каждого автомата только по одной многозначной переменной;

– отделить представление входных и выходных воздействий от их содержания;

– обеспечить протоколирование **в терминах автоматов** с целью проверки корректности построенной программы, представляющей собой систему взаимосвязанных объектов.

Необходимо отметить, что единственное ограничение, накладываемое на подход, предложенный в работах [6,10] для событийных систем, при его применении в объектно-ориентированном программировании, заключается в том, что для сохранения принципа инкапсуляции обмен номерами состояний должен производиться только между автоматами, принадлежащими одному классу.

ПРЕДЛАГАЕМЫЙ ПОДХОД

Также как и при использовании любого другого подхода, применение предлагаемого связано со множеством эвристик, возвратов назад, уточнений и параллельно выполняемых работ. Однако, после завершения создания программы, предлагаемый подход может быть сформулирован (по крайней мере для полного ее документирования) как "идеальная" технология, фиксирующая принятые решения.

1. На основе анализа предметной области выделяются классы и строится диаграмма классов, отражающая, в основном, наследование и агрегирование (например, вложенность).

2. Для каждого класса разрабатывается словесное описание, по крайней мере, в форме перечня решаемых задач.

3. Для каждого класса создается его структурная схема, несколько напоминающая карту CRC (Class-Responsibility-Collaboration) [29]. Нотация, используемая при построении структурных схем классов, приведена на рис. 1. Отметим, что если класс не знает о вызывающих его объектах, то они не указываются. В случае наличия большого количества вызываемых объектов достаточно указать только основные из них.

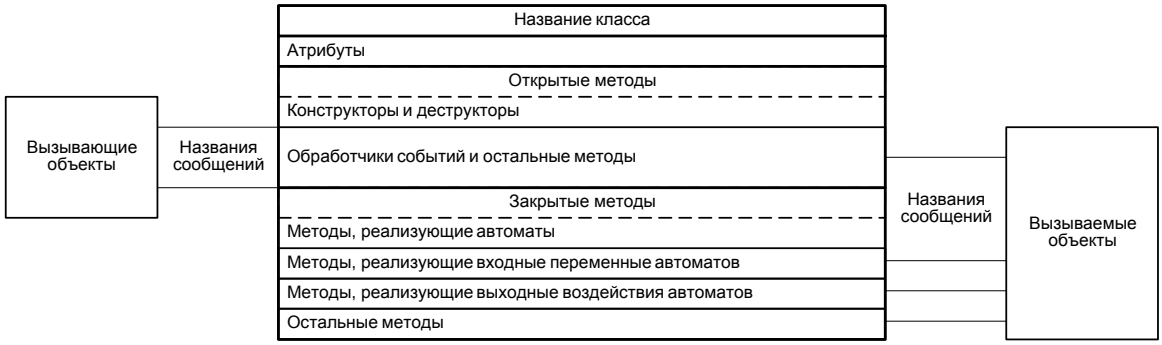

Рис. 1. Нотация, используемая при построении структурных схем классов

Интерфейс класса образуют открытые (public) атрибуты и методы, к которым обращаются другие объекты. Эти методы, в свою очередь, вызывают закрытые (private) методы рассматриваемого класса.

Как открытые, так и закрытые методы, в общем случае могут передавать сообщения другим объектам.

Закрытые методы могут быть разделены на две части: автоматные и остальные. Автоматные методы в общем случае делятся на три разновидности: реализующие автоматы; реализующие входные переменные автоматов; реализующие выходные воздействия автоматов.

4. При наличии в классе нескольких автоматов строится схема их взаимодействия [10].

5. Для каждого автомата разрабатывается словесное описание.

6. Для каждого автомата, используя правила, изложенные в работах [6,10], строится схема связей, определяющая его интерфейс. Входными воздействиями автомата являются входные переменные, предикаты, проверяющие номера состояний, и события. В схеме связей в качестве событий (наряду с другими) могут быть указаны сообщения, получаемые объектом и приведенные в структурной схеме класса.

В схеме связей показываются все события, с которыми автомат запускается, вне зависимости от того, используются ли они в пометках дуг и петель графа переходов. Для событий, не участвующих в условиях переходов, может приводиться соответствующий комментарий.

В отличие от структурной схемы класса, на входах и выходах в схеме связей автомата указываются не названия сообщений, а названия соответствующих входных и выходных воздействий.

7. Для каждого автомата на основе нотации, приведенной в работах [6,10], строится граф переходов.

8. Каждый класс реализуется соответствующим модулем программы. Его структура должна быть изоморфна структурной схеме класса, а методы, реализующие автоматы, строятся по шаблону, приведенному в работах [6,10]. При этом методы, соответствующие входным переменным и выходным воздействиям автомата, располагаются отдельно от метода, реализующего автомат, из которого они только вызываются.

Благодаря такому подходу, методы, соответствующие входным переменным и выходным воздействиям автомата, могут быть виртуальными (virtual), и переопределяться в порождаемых классах. Таким образом, имеется возможность создать базовый класс для управления некоторой группой "устройств", в котором реализуются только автоматы, а используемые ими входные переменные и выходные воздействия переопределяются на уровне порождаемых классов, управляющих конкретными "устройствами". Методы, реализующие автоматы, также могут переопределяться в порождаемых классах. Это обеспечивает построение классов, реализующих различные алгоритмы управления одними и теми же "устройствами".

9. Для изучения поведения программы, определяемого, в том числе, и взаимодействием объектов, а в ходе разработки — для ее отладки, **автоматически** строятся протоколы, описывающие работу всех автоматов в терминах состояний, переходов, событий, входных переменных и выходных воздействий с указанием соответствующих объектов. Это обеспечивается за счет включения функций протоколирования в соответствующие методы. При необходимости могут протоколироваться также и аналоговые параметры. Тот факт, что входные и выходные воздействия "привязаны" к объектам, автоматам и состояниям упрощает понимание таких протоколов по сравнению с протоколами, которые строятся традиционно. Из рассмотрения протоколов следует, что автоматы (в отличие от классов) абстракциями не являются. При этом можно утверждать, что автоматы структурируют поведение, которое с их помощью описывается весьма компактно и строго.

10. Выпускается созданная в ходе проекта документация.

Отметим, что в изложенном подходе реализация как классов, так и автоматов осуществляется строго в соответствии с диаграммами, полученными на этапе проектирования, и не допускает свободного их толкования. Предлагаемый подход не предполагает выполнение реализации "по мотивам" этих диаграмм. Реализация должна быть выполнена по диаграммам формально и изоморфно.

ТАНКИ

Рассматриваемый пример выбран из области создания виртуальных роботов, обладающих средствами обнаружения и уничтожения противника. Каждая из игр этого класса имеет специфику, определяемую средой исполнения, правилами проведения боев и используемым языком программирования.

Исходя из целей настоящей работы, авторов интересовала игра, которая в качестве языка программирования использовала бы не специализированный язык, а один из широко распространенных объектно-ориентированных языков. Единственной известной авторам игрой (в 2001 году), отвечающей этому требованию, являлась разработанная компанией "Alphaworks" игра "Robocode" (**http://robocode.alphaworks.ibm.com**). В 2002 году эта ситуация несколько изменилась: другая версия этой игры была предложена в качестве "разминки" для участников командного студенческого чемпионата мира по программированию ACM. В 2003 году на предварительном туре была предложена игра "CodeRally", принадлежащая к этому же классу. К играм того же класса относится и разработанная фирмой Microsoft игра "Terrarium" (**http://www.terrariumgame.net**).

Игра "Robocode" позиционируется фирмой IBM как средство для обучения программированию на языке Java на примере создания программы, являющейся системой управления танком, предоставляемым средой исполнения. Отметим, что использование интерпретируемого языка в играх данного жанра дает возможность легко и быстро внедрять разработанную программу. На рис. 2 приведен внешний вид среды исполнения в процессе боя двух танков.

Рис. 2. Фрагмент боя

РЕЗУЛЬТАТЫ СРАЖЕНИЙ

История создания танка состоит из двух частей. Сначала был создан **нестреляющий** танк (counterwallrobot.Cynical_1), который в течении нескольких дней (с 28.09.2001 по 02.10.2001) и **был лучшим** на открытом турнире, практически ежедневно проводимом создателем сайта **http://robocode.isbeautiful.org**. Начиная с 15.10.01, он стал **стреляющим** (counterwallrobot.Cynical_2), и занимал 5–6 место в мире. Разработка следующей версии (counterwallrobot.Cynical_3) была приостановлена, так как он выполнил свое предназначение применительно к разработке предлагаемой технологии. Несмотря на это, он все еще находится в десятке лучших танков мира.

Созданный танк отмечен в работе [30] как "вызвавший наибольший интерес", поскольку построен "по науке" и является, по-видимому, единственным, для которого была выпущена проектная документация. Эта документация для последней версии (Cynical_3) танка, разработанная в соответствии с предлагаемой технологией, размещена на сайте **http://is.ifmo.ru** в разделе "Проекты".

ПРОЕКТИРОВАНИЕ ПРОГРАММЫ

Изложим процесс создания программы управления танком в соответствии с предлагаемой технологией.

1. Первым разрабатываемым документом является диаграмма классов, в верхней части которой расположены предоставляемые разработчику классы (в том числе из стандартной библиотеки) и среда исполнения (рис. 3).

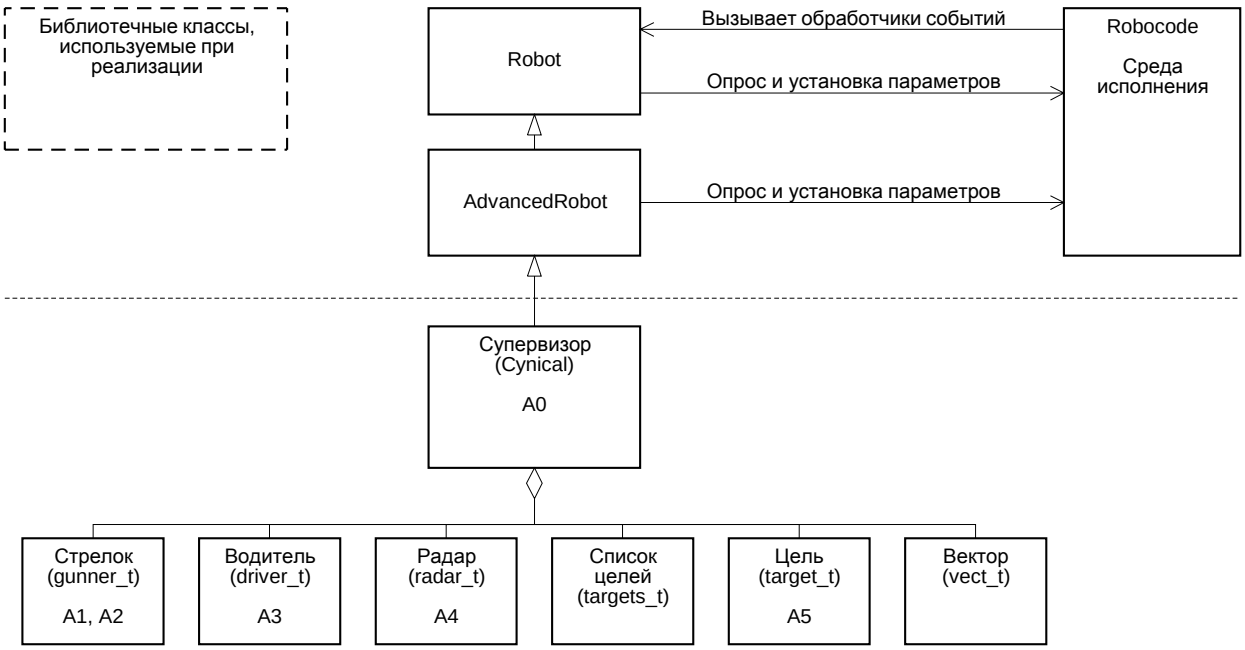

Рис. 3. Диаграмма классов

Среда формирует определенные правилами игры события и вызывает их обработчики, объявленные в классе "Robot". Указанные обработчики наследуются в классе "AdvancedRobot". Эти классы реализуют вызываемые из разрабатываемой системы управления методы, обеспечивающие непосредственное управление танком за счет опроса и установки его параметров в среде исполнения. К такой разновидности методов относится, например, метод turnRight(), который поворачивает танк вправо.

В соответствии с требованиями программного интерфейса среды исполнения к структуре программы, она должна представлять собой один класс, порожденный от класса "Robot" или, как в данном случае, от класса "AdvancedRobot". Обработчики событий, объявленные в классах "Robot" и "AdvancedRobot", определяются в порожденном от них классе "Cynical".

Название содержащего программу пакета "counterwallrobot" вместе с названием головного класса образуют название танка. Головной класс "Супервизор" в программе имеет имя "Cynical", и содержит шесть внутренних (вложенных) классов, сформированных на основе анализа задачи, которые носят следующие названия: "Стрелок", "Водитель", "Радар", "Список целей", "Цель" и "Вектор". Класс "Стрелок" является системой управления стрельбой, класс "Водитель" – системой управления маневрированием, а класс "Радар" – системой управления радаром.

Все эти шесть классов реализованы как внутренние для того, чтобы они не воспринимались средой исполнения как самостоятельные системы управления танками.

На диаграмме классы, содержащие автоматы, имеют соответствующие пометки.

Проектирование классов, рассмотрим на примере класса "Стрелок", содержащего автоматы A1 и A2.

2. Этот класс предназначен для управления стрельбой и решает следующие задачи:

– выбор цели;

– анализ параметров цели и расчет упреждения;

– управление пушкой;

– определение скорости охлаждения пушки.

3. Структурная схема класса "Стрелок" приведена на рис. 4.

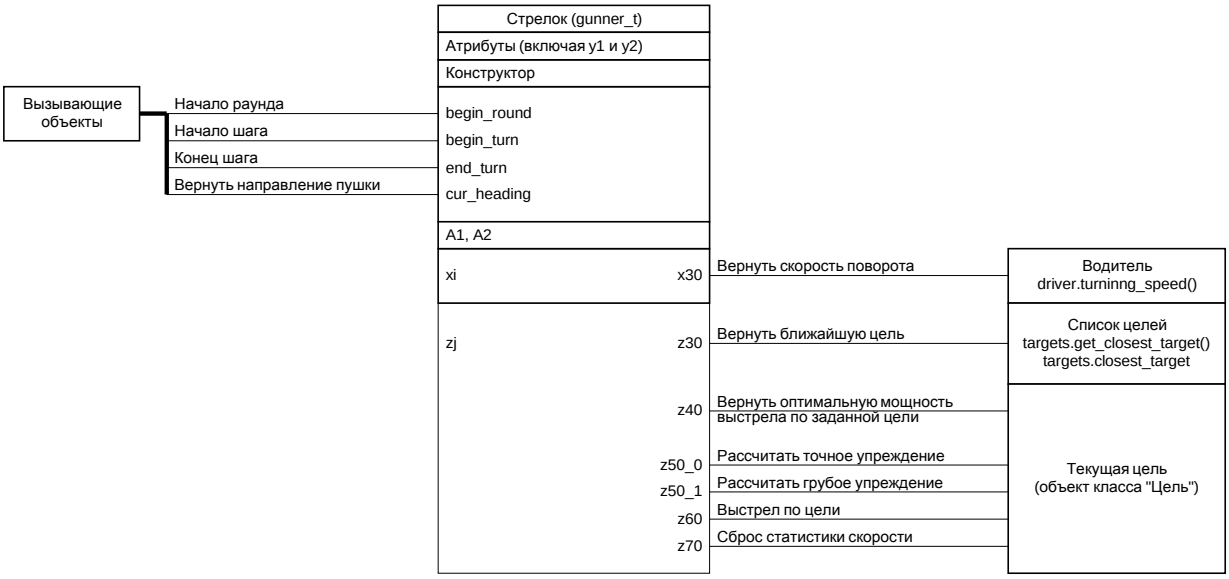

Рис. 4. Структурная схема класса "Стрелок"

Из рассмотрения структурной схемы следует, что в этом классе закрытые методы, отличные от автоматных, отсутствуют.

Интерфейс этого класса состоит из трех обработчиков событий ("Начало раунда", "Начало шага" и "Конец шага") и метода, возвращающего вызвавшему его объекту текущее направление пушки.

Закрытая часть класса состоит из атрибутов и методов, реализующих автоматы A1, A2 и их входные переменные и выходные воздействия. В этой схеме указаны обозначения только тех методов, реализующих входные переменные и выходные воздействия, которые передают сообщения другим объектам. Остальные автоматные методы (например, z80) указываются в схемах связей автоматов.

4. Так как класс содержит более одного автомата, то строится схема их взаимодействия (рис. 5), являющаяся, в данном случае (например, в отличие от приведенной в работе [10]), весьма простой.

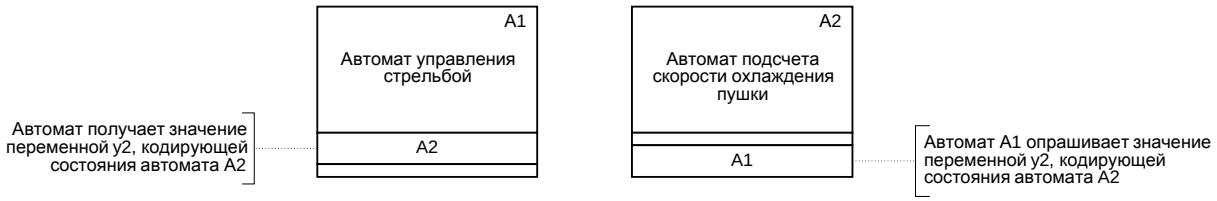

Рис. 5. Схема взаимодействия автоматов класса "Стрелок"

Проектирование автоматов, входящих в класс, рассмотрим на примере автомата A1, участвующего в решении первых трех из указанных выше задач.

5. В выполнении данного этапа нет необходимости.

6. Схема связей автомата A1 приведена на рис. 6. Отметим, что в этой схеме, в отличие от структурной схемы класса, на линиях указываются не названия сообщений, а названия входных и выходных воздействий. Например, выходное воздействие z30, названное на рис. 6 "Выбрать цель", осуществляя выбор цели, передает объекту класса "Список целей" сообщение "Вернуть ближайшую цель" (рис. 4).

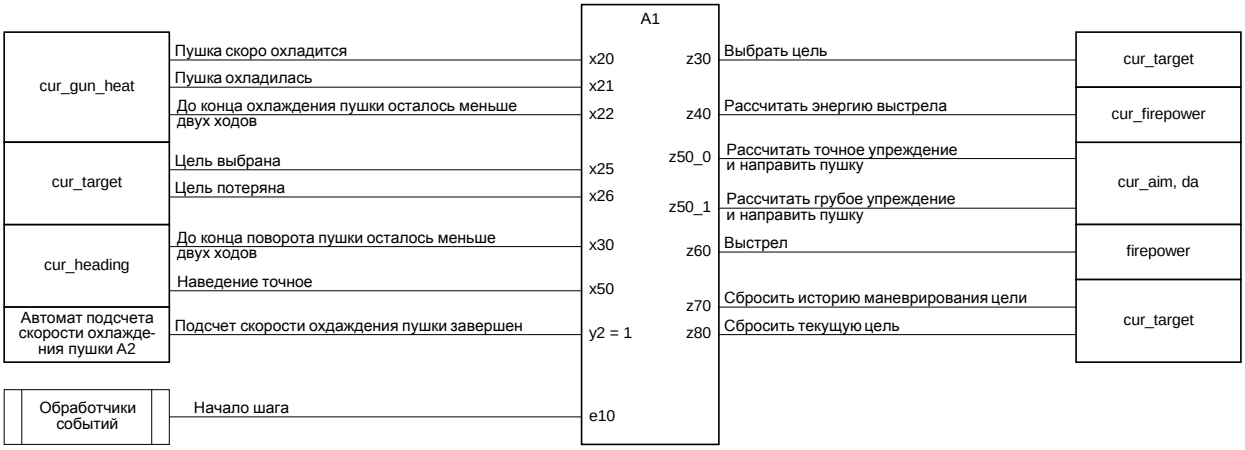

Рис. 6. Схема связей автомата управления стрельбой

7. На рис. 7 приведен граф переходов автомата A1.

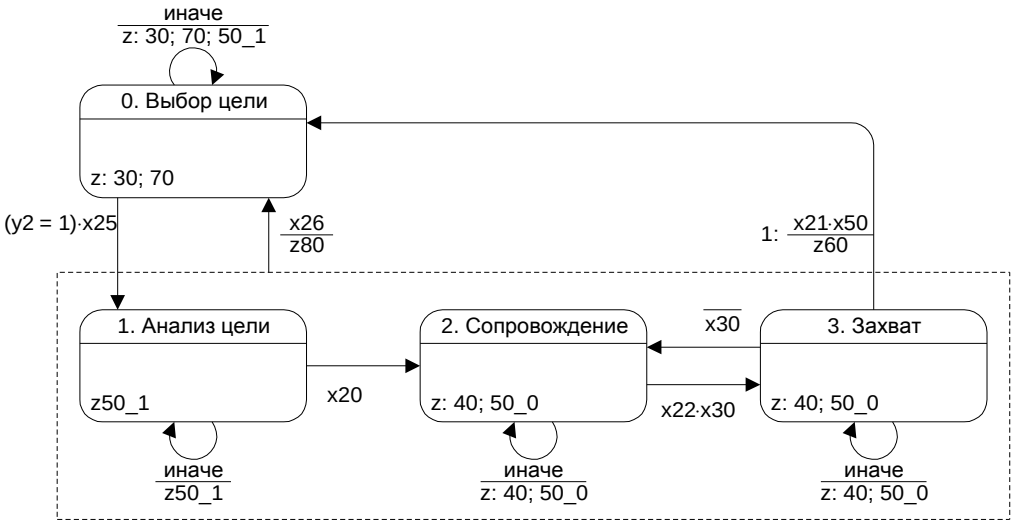

Рис. 7. Граф переходов автомата управления стрельбой

8. Листинг 1 содержит фрагмент программы, реализующий рассматриваемый класс. Структура этого фрагмента определяется схемой, приведенной на рис. 4.

ЛИСТИНГ 1. Фрагмент программы, реализующей класс "Стрелок"

```
public class gunner_t extends Object
{
//*** Атрибуты.
  ...
  private int           y1 = 0 ;
  private int           y2 = 0 ;

//*** Открытые методы.
  // Конструктор: gunner_t().

  // Начало раунда: begin_round().

  // Начало шага.
  public void
  begin_turn()
  {
    old_heading = cur_heading ;
    cur_heading = getGunHeadingRadians() ;
    old_gun_heat = cur_gun_heat ;
    cur_gun_heat = getGunHeat() ;

    da = 0 ;
    firepower = 0 ;
```

```
    A1(10) ;
    A2(10) ;
  }

  // Конец шага: end_turn().

  // Вернуть направление пушки: cur_heading().

//*** Закрытые методы.
  // Автомат A1.
  private void
  A1( int e )
  {
    int   y_old = y1 ;

    if( OBJECTS_LOGGING )
      log( "Для объекта 'Стрелок':" ) ;

    if( A1_BEGIN_LOGGING )
      log_begin( "A1", y1, e ) ;

    switch( y1 )
    {
      case 0:
        if( (y2 == 1) && x25() )                        y1 = 1 ;
        else
                              { z30() ; z70() ; z50_1() ; }
      break ;

      case 1:
        if( x26() )           { z80() ;                 y1 = 0 ; }
        else
        if( x20() )                                     y1 = 2 ;
        else
                              { z50_1() ; }
      break ;

      case 2:
        if( x26() )           { z80() ;                 y1 = 0 ; }
        else
        if( x22() && x30() )                            y1 = 3 ;
        else
                              { z40() ; z50_0() ; }
      break ;

      case 3:
        if( x26() )           { z80() ;                 y1 = 0 ; }
        else
        if( x21() && x50() ) { z60() ;                  y1 = 0 ; }
        else
        if( !x30() )                                    y1 = 2 ;
        else
                              { z40() ; z50_0() ; }
      break ;

      default :
        if( A1_ERROR_LOGGING )
          log_error( "A1", y1 ) ;
    }

    if( y1 != y_old )
    {

    if( A1_TRANS_LOGGING )
      log_trans( "A1", y1, y_old ) ;

    switch( y1 )
    {
      case 0:
        z30() ; z70() ;
      break ;

      case 1:
        z50_1() ;
      break ;
```

```
      case 2:
        z40() ; z50_0() ;
      break ;

      case 3:
        z40() ; z50_0() ;
      break ;
    }
    }

    if( A1_END_LOGGING )
      log_end( "A1", y1 ) ;
  }

  // Автомат A2.
  ...

//*** Реализация входных переменных.

  // x10: Подсчет скорости охлаждения пушки завершен. Автомат A2.

  private boolean
  x20()
  {
    boolean result = cur_gun_heat/gun_heat_decrement <= 3 ;

    if( INPUTS_LOGGING )
      log_input( "x20", "Пушка скоро охладится", result ) ;
    return result ;
  }

  // x21: Пушка охладилась.

  // x22: До конца охлаждения пушки меньше двух ходов.

  // x25: Цель выбрана.

  // x26: Цель потеряна.

  private boolean
  x30()
  {
    boolean result = true ;

    double  gun_to_go = get_angle_diff( cur_heading, cur_aim.a ) ;
    double  turn_direction = gun_to_go >= 0 ? 1 : -1 ;
                              // Обращение к объекту класса "Водитель".
    double  gun_turning_speed = turn_direction * gun_rotation_speed
                             + driver.turning_speed() ;

    result = Math.abs( gun_to_go / gun_turning_speed ) <= 1 ;

    if( INPUTS_LOGGING )
      log_input( "x30", "До конца поворота пушки меньше двух ходов", result ) ;
    return result ;
  }

  // x50: Наведение точное.

//*** Реализация выходных воздействий.
  private void
  z30()
  {
    if( OUTPUTS_LOGGING )
      log_output( "z30", "Выбрать цель" ) ;

    cur_target = targets.get_closest_target( 8 ) ;
    if( cur_target == null )
      cur_target = targets.closest_target ;
  }

  // z35: Подсчет скорости охлаждения пушки. Автомат A2.

  // z36: Вывести скорость охлаждения пушки. Автомат A2.
```

```
  // z40: Рассчитать мощность выстрела.

  // z50_0: Рассчитать точное упреждение и направить пушку.

  // z50_1: Рассчитать грубое упреждение и направить пушку.

  // z60: Выстрел.

  // z70: Сбросить историю маневрирования цели.

  // z80: Сбросить текущую цель.

} // gunner_t
```

9. Листинг 2 содержит фрагмент протокола функционирования разработанной системы управления для одного раунда боя двух танков.

Тип записи в протоколе обозначается одним из следующих символов: '{' — запуск автомата, '}' — завершение работы автомата, 'T' — переход в автомате (от англ. "transition"), 'i' — опрос входной переменной, '*' — формирование выходного воздействия.

Для выделения действий, выполняемых внутри автоматов, используются отступы. При применении вложенных автоматов отступы используются также и для обозначения уровней вложенности.

ЛИСТИНГ 2. Фрагмент протокола боя двух танков

```
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 0 с событием e9
 * z10_0: Инициализация при запуске.
 T A0: Автомат A0 перешел из состояния 0 в состояние 1
 * z10_1: Инициализация в начале раунда.

*** Раунд 1

 * z10_2: Инициализация в начале шага.
} A0: Автомат A0 завершил свою работу в состоянии 1
--------- 0 --------- Начальный шаг (e9)
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага.
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 0 с событием e10
 * z30: Выбрать цель.
 * z70: Сбросить историю маневрирования цели.
 * z50_1: Рассчитать грубое упреждение и направить пушку.
} A1: Автомат A1 завершил свою работу в состоянии 0
{ A2: Автомат A2 запущен в состоянии 0 с событием e10
 i x10: Подсчет скорости охлаждения пушки завершен? - НЕТ.
 * z35: Подсчет скорости охлаждения пушки.
} A2: Автомат A2 завершил свою работу в состоянии 0
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 0 с событием e10
 i x70: Цикл сканирования завершен? - НЕТ.
 * z100_0: Повернуть радар влево.
} A4: Автомат A4 завершил свою работу в состоянии 0
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА.
 i x110: Сработал таймер T110? - ДА.
 * z200_0: Инициализация движения по траектории 'Маятник'.
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'.
 * z200_2: Определить направление и скорость движения 'Маятник'.
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 30 --------- Выстрел по цели
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели.
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
```

```
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага.
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ.
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 3 с событием e10
 i x26: Цель потеряна? - НЕТ.
 i x21: Пушка охладилась? - ДА.
 i x50: Наведение точное? - ДА.
 * z60: Выстрел.
 T A1: Автомат A1 перешел из состояния 3 в состояние 0
 * z30: Выбрать цель.
 * z70: Сбросить историю маневрирования цели.
} A1: Автомат A1 завершил свою работу в состоянии 0
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 0 с событием e10
 i x70: Цикл сканирования завершен? - ДА.
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА.
 * z101_0: Сбросить память пройденного радаром пути.
 T A4: Автомат A4 перешел из состояния 0 в состояние 1
 * z100_1: Повернуть радар вправо.
} A4: Автомат A4 завершил свою работу в состоянии 1
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА.
 i x110: Сработал таймер T110? - ДА.
 * z200_0: Инициализация движения по траектории 'Маятник'.
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'.
 * z200_2: Определить направление и скорость движения 'Маятник'.
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 42 --------- Попадание в цель (e130)
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели.
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e130
 * z1010: Обновить статистику попаданий в цель.
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага.
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ.
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 1 с событием e10
 i x26: Цель потеряна? - НЕТ.
 i x20: Пушка скоро охладится? - НЕТ.
 * z50_1: Рассчитать грубое упреждение и направить пушку.
} A1: Автомат A1 завершил свою работу в состоянии 1
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 1 с событием e10
 i x70: Цикл сканирования завершен? - ДА.
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА.
 * z101_0: Сбросить память пройденного радаром пути.
 T A4: Автомат A4 перешел из состояния 1 в состояние 0
 * z100_0: Повернуть радар влево.
} A4: Автомат A4 завершил свою работу в состоянии 0
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА.
 i x110: Сработал таймер T110? - ДА.
 * z200_0: Инициализация движения по траектории 'Маятник'.
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'.
 * z200_2: Определить направление и скорость движения 'Маятник'.
} A3: Автомат A3 завершил свою работу в состоянии 0
```

```
--------- 59 --------- Попадание в нас (e45)
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели.
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e45
 i x100: Враг близко? - ДА.
 i x110: Сработал таймер T110? - ДА.
 * z200_0: Инициализация движения по траектории 'Маятник'.
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'.
 * z200_2: Определить направление и скорость движения 'Маятник'.
} A3: Автомат A3 завершил свою работу в состоянии 0
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e136
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага.
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ.
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 2 с событием e10
 i x26: Цель потеряна? - НЕТ.
 i x30: До конца поворота пушки меньше двух ходов? - ДА.
 i x22: До конца охлаждения пушки меньше двух ходов? - ДА.
 T A1: Автомат A1 перешел из состояния 2 в состояние 3
 * z40: Рассчитать мощность выстрела.
 * z50_0: Рассчитать точное упреждение и направить пушку.
} A1: Автомат A1 завершил свою работу в состоянии 3
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 0 с событием e10
 i x70: Цикл сканирования завершен? - ДА.
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА.
 * z101_0: Сбросить память пройденного радаром пути.
 T A4: Автомат A4 перешел из состояния 0 в состояние 1
 * z100_1: Повернуть радар вправо.
} A4: Автомат A4 завершил свою работу в состоянии 1
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА.
 i x110: Сработал таймер T110? - ДА.
 * z200_0: Инициализация движения по траектории 'Маятник'.
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'.
 * z200_2: Определить направление и скорость движения 'Маятник'.
} A3: Автомат A3 завершил свою работу в состоянии 0

--------- 332 --------- Конец раунда (e20)
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e100
 * z1001: Обновить параметры цели.
} A5: Автомат A5 завершил свою работу в состоянии 2
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e10
 * z10_2: Инициализация в начале шага.
Для объекта 'Цель' (gg.Tarsier):
{ A5: Автомат A5 запущен в состоянии 2 с событием e140
 i x1000: Информация о цели устарела? - НЕТ.
} A5: Автомат A5 завершил свою работу в состоянии 2
} A0: Автомат A0 завершил свою работу в состоянии 1
Для объекта 'Стрелок':
{ A1: Автомат A1 запущен в состоянии 0 с событием e10
 i x25: Цель выбрана? - ДА.
 T A1: Автомат A1 перешел из состояния 0 в состояние 1
 * z50_1: Рассчитать грубое упреждение и направить пушку.
} A1: Автомат A1 завершил свою работу в состоянии 1
{ A2: Автомат A2 запущен в состоянии 1 с событием e10
} A2: Автомат A2 завершил свою работу в состоянии 1
Для объекта 'Радар':
{ A4: Автомат A4 запущен в состоянии 1 с событием e10
```

```
 i x70: Цикл сканирования завершен? - ДА.
 i x80: Пройденный радаром путь меньше 180 градусов? - ДА.
 * z101_0: Сбросить память пройденного радаром пути.
 T A4: Автомат A4 перешел из состояния 1 в состояние 0
 * z100_0: Повернуть радар влево.
} A4: Автомат A4 завершил свою работу в состоянии 0
Для объекта 'Водитель':
{ A3: Автомат A3 запущен в состоянии 0 с событием e10
 i x100: Враг близко? - ДА.
 i x110: Сработал таймер T110? - ДА.
 * z200_0: Инициализация движения по траектории 'Маятник'.
 * z200_1: Добавить случайную составляющую к траектории 'Маятник'.
 * z200_2: Определить направление и скорость движения 'Маятник'.
} A3: Автомат A3 завершил свою работу в состоянии 0
Для объекта 'Супервизор':
{ A0: Автомат A0 запущен в состоянии 1 с событием e20
 T A0: Автомат A0 перешел из состояния 1 в состояние 2
 * z20: Вывести статистику раунда.
---- Статистика для gg.Tarsier ----
Выстрелов: 23, попаданий: 5
Вероятность: 0.217, базовая: 0.943
---------
Выстрелов: 23, попаданий: 5, промахов: 18
Меткость: 0.227
Попали в нас: 7
Столкновений со стенами: 0
} A0: Автомат A0 завершил свою работу в состоянии 2
```

Как было отмечено выше, при необходимости протоколы могут быть дополнены информацией о значениях аналоговых параметров. Например, при выполнении функции, реализующей выходное воздействие, определяющее необходимое направление пушки (z50_0), дополнительно может выводиться полученный результат.

Отметим также, что если в игре участвуют два "своих" танка, то может быть запротоколирован весь бой, включая все попадания снарядами и гибель по крайней мере одного из них. Это позволяет полностью восстановить всю историю боя, а при необходимости понять причины принятия тех или иных решений.

10. Документация на программу в целом, разработанная на основе предлагаемого подхода, приведена на сайте **http://is.ifmo.ru** в разделе "Проекты".

В заключение раздела отметим, что использование состояний, кроме указанных выше преимуществ, уменьшает вероятность того, что программа не успеет выполнить необходимые расчеты в отведенный ей интервал времени. Так, например, из рис. 7 следует, что точный расчет упреждения (z50_0), занимающий значительное время, выполняется только в состояниях 2 и 3, в которых автомат находится гораздо меньшее время, чем в остальных состояниях, в которых расчет упреждения выполняется грубо (z50_1).

СРАВНЕНИЕ С UML

Перечислим имеющиеся, по нашему мнению, недостатки *UML*, являющегося в настоящее время наиболее известным и распространенным в мире графическим языком для визуализации, спецификации, конструирования и документирования систем, программное обеспечение которых разрабатывается на основе объектного подхода:

– неудачное название языка, так как он предназначен "не для моделирования как такового" [8], а для создания моделей в виде диаграмм, позволяющих описывать различные "стороны" программного обеспечения;

– является языком, позволяющим описывать различные модели, а не технологией разработки на их основе программного обеспечения;

– применяются весьма странные термины, например, "событие-триггер" и "сторожевое условие", в то время как в вычислительной технике термины "событие", "триггер" и "условие" имеют вполне определенный и традиционный смысл;

– необоснованно много внимания уделяется диаграммам прецедентов (диаграммам использования), в которых "прецедент (Use case) специфицирует поведение системы или ее части и представляет собой описание множества последовательностей действий (включая варианты), выполняемых системой (совокупностью образующих ее объектов) для того, чтобы актер мог получить определенный результат" [8]. При сложном поведении рассматриваемых компонент построить на этапе проектирования исчерпывающие диаграммы такого типа практически невозможно. Обратим внимание, что в рассмотренном примере вообще нет актера (действующего лица), который хотел бы получить результат, отличный от простого выполнения танком требований среды разработки, и поэтому в данном случае отсутствует возможность построения рассматриваемой разновидности диаграмм;

– диаграммы кооперации при большом числе сообщений у взаимодействующих объектов становятся необозримыми;

– при сложном поведении объектов построить исчерпывающие диаграммы последовательностей на этапе проектирования практически невозможно;

– каждая диаграмма состояний предназначена "для моделирования жизненного цикла объекта" [8]. Поэтому при построении таких диаграмм могут использоваться вложенные состояния, но не вложенные автоматы. Действительно, если применять термин "жизненный цикл", то без учета реинкарнации одному объекту может соответствовать только одна диаграмма состояний. Делая акцент на поведении объекта, а не на его жизненном цикле, вопрос об единственности автомата для каждого объекта снимается. У объекта может быть много методов, часть из которых (а не только один) могут быть автоматными. При такой трактовке использование автоматов в объектном программировании становится более понятным по сравнению с приведенными выше определениями "пророков";

– использование словесных, а не символьных пометок дуг, петель и вершин при построении диаграмм состояний, не позволяет получать эти диаграммы в строгом и компактном виде, особенно для объектов, обладающих сложным поведением;

– если в некоторых книгах, например в [8], еще упоминаются такие термины, как "автомат Мура" или "автомат Мили", то уже в стандарте [31], эти термины исключены, в то время как в предлагаемом подходе [12] используемый тип модели автомата определяет его реализацию;

– ввиду того, что в работах [12,32] показано, что последовательно-параллельные процессы могут быть описаны системой взаимосвязанных графов переходов, то в применении диаграмм деятельностей нет необходимости;

– вычислительные алгоритмы могут быть описаны не с помощью диаграмм деятельностей, а на основе традиционных схем алгоритмов, которые при необходимости формально могут быть преобразованы в графы переходов [12].

Из изложенного следует, что при создании языка *UML* был нарушен "принцип Оккама", так как в него введено много сущностей и терминов без необходимости. Кстати, такого же мнения придерживаемся не только мы. Говоря о книге "трех друзей" [33], автор работы [2] пишет: "Перед тем как начать читать эту книгу я приготовился к самому худшему. Я был приятно удивлен — лишь некоторые места книги содержали объяснения, которые были написаны так, будто сами авторы книги не совсем понимали, о чем речь". Также, он пишет: "когда вы в первый раз сталкиваетесь с языком *UML*, кажется, что его невозможно понять — такое там множество диаграмм и мелочей. Во втором издании книги [29] ее авторы утверждают, что большая часть всего этого попросту никому не нужна, и поэтому они

отбрасывают малозначительные детали и говорят только о самом главном из имеющегося в языке".

Авторы в настоящей работе пошли дальше:

– стараясь приблизиться к принципам построения "легких" методологий [34] исключили диаграммы прецедентов, кооперации, последовательностей и действий;

– для каждого класса ввели его структурную схему и нотацию для ее построения;

– для каждого класса, содержащего более одного автомата, ввели схему взаимодействия автоматов;

– для каждого автомата ввели схему связей;

– изменили нотацию диаграмм состояний, например, в части введения в их вершины вложенных автоматов;

– ввели шаблон для стандартной реализации каждого автомата;

– вместо построения сценариев при создании программы, ввели исчерпывающее **автоматическое** протоколирование, выполняемое в терминах автоматов при ее отладке и сертификации. Такое протоколирование может быть полезно при построении **черных** (аварийных) **ящиков**, обеспечивая при необходимости фиксацию **всего** поведения программы. Это особенно важно в связи с тем, что главная проблема проектирования состоит не в обеспечении безошибочности, которую достичь невозможно, а в создании подходящих способов выяснения, "что именно пошло не так и как это исправить" [34]. Изложенное 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" [35]. Это 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" [27];

– предложили технологию создания программного обеспечения, предназначенную для совместного использования объектно-ориентированного и автоматного программирования.

СРАВНЕНИЕ ПРЕДЛОЖЕННОЙ РЕАЛИЗАЦИИ С "БОЛЕЕ ОБЪЕКТНОЙ"

Все большее количество программистов, с которыми авторы общаются, соглашаются с тем, что обработчики событий должны вызывать автоматы, а не содержать распределенную по ним логику, как это принято традиционно. Это связано с тем, что одна из важнейших целей, для достижения которой были введены объекты, состояла в упрощении внесения изменений при повторном использовании. Эта же цель достигается для логики при использовании автоматов. Поэтому совместное применение объектной и автоматной парадигм весьма увеличивает эффективность каждой из них.

При этом любая программа, построенная с одновременным использованием объектно-ориентированного и автоматного программирования может быть названа объектно-ориентированной с явным выделением состояний.

Однако по степени "объектно-ориентированности" реализации могут отличаться. В предложенной нами реализации в одном классе находятся как объект управления, так и управляющие им автоматы. Это напоминает живого человека, у которого душа и тело неотделимы. Такая реализация в системах управления называется "автоматизированный объект управления".

Многие считают эту реализацию недостаточно объектной, указывая на имеющее место "оборачивание" автоматов классами.

Другой подход состоит в том, что объект управления и управляющие им автоматы реализуются в отдельных классах. При этом возможны различные реализации, среди которых выделим паттерн "State" или его модификации [19]. Считается, что такие реализации в большей степени, чем единая конструкция `switch`, соответствуют объектному подходу.

Еще одним "недостатком" нашего подхода считается символическое (а не смысловое) обозначение переменных. Поэтому получаемая программа не является самодокументирующейся.

Для выяснения справедливости этих замечаний студент кафедры "Компьютерные технологии" СПбГИТМО (ТУ) Денис Кузнецов (двукратный призер студенческого командного чемпионата мира по программированию ACM) выполнил рефакторинг нашего проекта — изменил структуру кода без изменения его функциональности с целью учета указанных "недостатков".

Ниже приводятся приложения, содержащие два варианта реализации класса "Стрелок": предлагаемый авторами (приложение 1) и выполненный Д. Кузнецовым (приложение 2). В тексты программ добавлены комментарии, позволяющие проследить порядок вызова методов, выполняемых при запуске автомата A1. Каждый "этап" работы автомата обозначается меткой, которые нумеруются начиная с нуля.

Второй вариант реализации класса "Стрелок", предложенный Д. Кузнецовым, характеризуется следующими особенностями:

– каждое состояние каждого автомата реализуется отдельным объектом, порожденным от некоторого базового класса (использован подход, аналогичный паттерну "State");

– текущее состояние хранится не в виде номера, а в виде ссылки на объект, реализующий это состояние;

– проверка условий переходов и действия на дугах и петлях выполняются в методе `processIncomingEvent`;

– действия при входе в состояния выполняются в методе `onEnter`;

– использованы смысловые обозначения входных и выходных воздействий.

Из сравнения двух рассмотренных вариантов реализации можно сделать следующие выводы:

– в предложенном подходе логика работы программы централизована и сосредоточена внутри метода, реализующего автомат, что обеспечивает полное соответствие программы графу переходов;

– при втором подходе логика работы программы децентрализована и рассредоточена по нескольким классам. При этом теряется одно из основных преимуществ автоматного программирования, состоящее в указанном выше соответствии;

– из-за децентрализации программы она становится более сложной для понимания. Для сравнения вариантов в листинги введены метки, обозначающие этапы работы автомата A1. Количество этапов увеличилось по сравнению с предложенным подходом с 10 до 18, а

глубина вызовов с одного до четырех. Из-за этого программу очень трудно читать по листингу, так как чаще приходится "прыгать" с одного листа на другой;

– при втором подходе используются смысловые названия переменных, не соответствующие символьным обозначениям, использованным в схеме связей и на графе переходов. Из-за этого сложно проследить соответствие между графом переходов и текстом программы. В дальнейшем Д. Кузнецов предложил в качестве компромисса использовать "смешанные названия", состоящие из символьного и смыслового обозначения (**http://is.ifmo.ru**, раздел "Проекты").

ЗАКЛЮЧЕНИЕ

Завершая статью, отметим, что несмотря на большую важность объектно-ориентированного программирования и огромную его рекламу к этой парадигме все программирование не сводится. Во-первых, существуют другие парадигмы программирования на алгоритмических языках высокого уровня, а, во-вторых, широко используются вычислительные устройства, отличные от персональных компьютеров, такие как программируемые логические контроллеры, микроконтроллеры и микропроцессоры для встроенных систем, которые программируются на специализированных языках, на подмножестве языка Си, либо на языках низкого уровня. При этом, если еще недавно на долю процессоров для персональных компьютеров приходилось 6% от всех выпускаемых микропроцессоров, то в настоящее время их доля уже не превышает 2% [36], хотя объем программ, написанных для них, весьма значителен.

Многие из устройств, работающих под управлением оставшихся 98% микропроцессоров, являются управляющими и могут программироваться с **явным выделением состояний** [6], но не обязательно объектно, так как "объектность" обычно связана с избыточностью.

Отметим также, что понятие "состояние" является базовым практически во всех "развитых" науках, таких например, как механика, физика, металлургия, теория управления, теория автоматов. Более того, Нобелевскую премию по физике за 2001 год получили Э. Корнел, К. Виман и В. Кеттерле, которые дополнительно к четырем известным состояниям вещества, открыли (теоретически предсказанное Ш. Бозе и А. Эйнштейном еще в 1924 году) пятое состояние — сверхконденсированное, возникающее при сверхнизких температурах, когда атомы теряют свою самостоятельность, что приводит к резкому изменению всех свойств вещества.

Однако такая ситуация характерна не для всех наук. Например в такой науке, как история, важнейшим является понятие "событие", которое в настоящее время является определяющим и в практическом программировании при использовании сред быстрой разработки программ, называемых также событийно-ориентированными [37].

Подход, излагаемый в настоящей работе, является продолжением работ авторов, направленных на широкое внедрение таких понятий, как "состояние" и "автомат", в инженерное программирование (Software Engineering). Он берет начало в теории автоматов и в проектировании систем управления и не ставит своей целью учесть все особенности объектных языков программирования, которые описаны в таких книгах, как [1] и [2]. Настоящая работа является первой редакцией предлагаемого подхода для рассматриваемого класса задач и в дальнейшем будет усовершенствована. В частности, автоматы могут реализовываться не в виде методов класса, как в настоящей работе, а отдельными классами. Кроме того, каждое состояние может рассматриваться как отдельный объект [19].

Из изложенного следует, что объектно-ориентированное программирование с явным выделением состояний базируется на двух парадигмах: объектной и автоматной. Это

соответствует идее использования нескольких парадигм, развиваемой в настоящее время в программировании [38]. При этом, если "объектный подход предоставляет программисту средства для решения задачи в ее пространстве" [2], то автоматный подход — средства для описания поведения объектов в терминах пространства состояний. Применение автоматов проясняет поведение программы, также, как применение объектов проясняет ее структуру.

Авторы надеются, что предложенный подход внесет определенный вклад в инженерное программирование, "в необходимости которого, по словам Г. Буча, многие сомневаются", а "Б. Гейтс публично заявляет, что не верит в диаграммы и не желает, чтобы его программисты занимались проектированием" [34]. И это при наличии огромного числа ошибок в выпускаемом программном обеспечении, которые в системах, управляющих ответственными объектами, недопустимы вовсе. Следует отметить, что в последнее время формальные методы (например, на основе абстрактных машин состояний при построении спецификаций [39,40]) начинают использоваться и в фирме "Microsoft". Однако "метод Гуревича", во-первых, требует от участников разработки знаний в области математической логики, а во-вторых, не охватывает все стадии создания программного обеспечения.

В заключение отметим, что подход, основанный на применении автоматов в программировании был использован и при создании других проектов. Некоторые из них приведены на сайте **http://is.ifmo.ru** в разделе "Проекты". В частности, в этом разделе приведены проекты систем управления "роботами" для указанных выше игр "CodeRally" и "Terrarium". Как и в случае игры "Robocode", для игры "Terrarium" созданы тысячи "существ", многие из которых распространяются с открытыми исходными текстами, однако проектная документация выпущена только для наших разработок.

Настоящая работа была выполнена на кафедре "Компьютерные технологии" Санкт-Петербургского государственного института точной механики и оптики (технического университета) при поддержке Российского фонда фундаментальных исследований по гранту №02-07-90114 "Разработка технологии автоматного программирования".

ЛИТЕРАТУРА

1. Страуструп Б. Язык программирования Си++. — М.: Бином, СПб.: Невский диалект, 1999.

2. Эккель Б. Философия Java. — СПб.: Питер, 2001.

3. Буч Г. Объектно-ориентированное проектирование с примерами применения. — Киев: Диалектика, М.: ИВК, 1992.

4. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. — М.: Бином; СПб.: Невский диалект, 1998.

5. Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. — Киев: Диалектика, 1993.

6. Шалыто А.А., Туккель Н.И. Программирование с явным выделением состояний // Мир ПК. — 2001. — № 8, 9.

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

8. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. — М.: ДМК, 2000.

9. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. — СПб.: Питер, 2001.

10. Шалыто А.А., Туккель Н.И. SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем // Программирование. — 2001. — № 5.

11. Шалыто А.А., Туккель Н.И. От тьюрингова программирования к автоматному программированию // Мир ПК. — 2001. — № 12.

12. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. — СПб.: Наука, 1998.

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

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

15. Martin R. Designing Object-Oriented C++ Applications Using the Booch Method. — NJ: Prentice Hall, 1993.

16. Любченко В.С. О бильярде с Microsoft Visial C++ 5.0 // Мир ПК. — 1998. — № 1.

17. Шлепнев А. Системно-ориентированное программирование // Мир ПК. — 2001. — № 6.

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

19. Гамма Э., Хелм Р., Джонсон Р. и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2001.

20. Ваганов С.А., Туккель Н.И., Шалыто А.А. Повышение централизации управления при программировании "реактивных" систем // Труды Международной научно-методической конференции "Телематика' 2001". — СПб.: СПбГИТМО (ТУ), 2001.

21. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и реактивных систем // Автоматика и телемеханика. — 2001. — № 1.

22. Дьяконов В. Simulink 4. Специальный справочник. — СПб.: Питер, 2002.

23. Фридман А.Л. Основы объектно-ориентированной разработки программных систем. — М.: Финансы и статистика, 2000.

24. Cardelli L., Wegner P. On Understanding Types, Data Abstraction and Polymorphism // ACM Computing Surveys. — 1985. — Vol. 17 (4). — December.

25. Jacobson I. et al. Object-Oriented Software Engineering. — NJ: Addison Wesley, 1992.

26. Бобровский С. Из пророков – в коммерсанты // PC WEEK/RE. — 4.02.1997.

27. Лавров С.С. Программирование. Математические основы, средства, теория. — СПб.: БХВ-Петербург, 2001.

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

29. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования. — М.: Мир, 1999.

30. Озеров А. Четыре танкиста и компьютер // Магия ПК. — 2002. — № 11.

31. UML Semantics. Version 1.0. — Rational Software Corp., 1997.

32. Odell J.J. Advanced object-oriented analysis & design using UML. — NY: SIGS Books, 1998.

33. Booch G., Rumbaugh J., Jacobson I. The Unified Software Development Process. — NJ: Addison-Wesley, 1999.

34. Программы следующего десятилетия // Открытые системы. — 2001. — №12.

35. Ершов А.П. Смешанные вычисления // В мире науки. — 1984. — № 6.

36. Петерсон Т. Там, где сходятся люди и машины // Computerworld Россия. — 21.08.2001.

37. Бобровский С. Самоучитель программирования на языке C++ в системе Borland C++ Builder 5.0. — М.: ДЕСС КОММ, 2001.

38. Budd T. Multiparadigm Programming in Leda. — NJ: Addison-Wesley, 1995.

39. Соловьев И.П. Формальные спецификации вычислительных систем. Машины абстрактных состояний (Машины Гуревича). — СПб.: СПбГУ, 1998.

40. Gurevich Y. et al. Using Abstract State Machines at Microsoft: A Case Study / Proceeding of ASM'2000 in "Abstract State Machines: Theory and Applications". — Lecture Notes in Computer Science. — 2000. — V.1912.

ПРИЛОЖЕНИЕ 1. ПРЕДЛАГАЕМАЯ РЕАЛИЗАЦИЯ КЛАССА "СТРЕЛОК"

```
//======
// Реализация класса "Стрелок" на основе автоматного программирования.
// Реализация выполнена Туккелем Н.И. и Шалыто А.А.
// В тексте приведен путь от запуска автомата до завершения его работы.
// Этапы этого пути обозначены пронумерованными метками.
// В этом пути учтены все выполняемые вызовы методов. Путь включает 10 этапов.
// Глубина вложенности вызовов не менее одного.
//

// Фрагмент файла Cynical_3.java, содержащий определение класса "Стрелок".
  public class gunner_t extends Object
  {
//*** Атрибуты.
    private final boolean SWITCH_DEBUG =      true ;
    private final boolean INPUTS_LOGGING =    true && SWITCH_DEBUG && ANY_DEBUG ;
    private final boolean OUTPUTS_LOGGING =   true && SWITCH_DEBUG && ANY_DEBUG ;
    private final boolean A1_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
    private final boolean A1_BEGIN_LOGGING =  true && A1_LOGGING && ANY_DEBUG ;
    private final boolean A1_END_LOGGING =    true && A1_LOGGING && ANY_DEBUG ;
    private final boolean A1_ERROR_LOGGING =  true && A1_LOGGING && ANY_DEBUG ;
    private final boolean A1_TRANS_LOGGING =  true && A1_LOGGING && ANY_DEBUG ;
    private final boolean A2_LOGGING =        true && SWITCH_DEBUG && ANY_DEBUG ;
    private final boolean A2_BEGIN_LOGGING =  true && A2_LOGGING && ANY_DEBUG ;
    private final boolean A2_END_LOGGING =    true && A2_LOGGING && ANY_DEBUG ;
    private final boolean A2_ERROR_LOGGING =  true && A2_LOGGING && ANY_DEBUG ;
    private final boolean A2_TRANS_LOGGING =  true && A2_LOGGING && ANY_DEBUG ;

    private double        old_heading, cur_heading ;  // Направление пушки.
    private double        old_gun_heat, cur_gun_heat, // Температура пушки.
                          gun_heat_decrement ;        // Скорость охлаждения пушки.
    private target_t      cur_target ;                // Текущая цель.
    private vect_t        cur_aim ;                   // Текущий прицел.
    private double        cur_firepower ;             // Текущая мощность выстрела.

    private double        da ;        // На сколько надо повернуть пушку на данном шаге.
    private double        firepower ;                 // Мощность производимого выстрела.

    private int           y1 = 0 ;
    private int           y2 = 0 ;

//*** Методы.

    // Конструктор.
    gunner_t()
    {
      gun_heat_decrement = 0 ;
      cur_aim = new vect_t() ;
    }

    // Начало раунда.
    public void
    begin_round()
    {
      old_heading = cur_heading = getGunHeadingRadians() ;
      old_gun_heat = cur_gun_heat = getGunHeat() ;
```

```
      cur_target = null ;
      cur_aim.reset() ;
      last_fire_time = 0 ;
    }

    // Начало шага.
    public void
    begin_turn()
    {
      old_heading = cur_heading ;
      cur_heading = getGunHeadingRadians() ;
      old_gun_heat = cur_gun_heat ;
      cur_gun_heat = getGunHeat() ;

      da = 0 ;
      firepower = 0 ;
```

/* Метка 0. Запуск автомата A1. Перейти на метку "1". */

```
      A1(10) ;
```

/* Метка 10. Завершение работы автомата A1. */

```
      A2(10) ;
    }

    // Конец шага.
    public void
    end_turn()
    {
      setTurnGunRightRadians( da ) ;

      //log( "firepower = " + firepower ) ;
      if( firepower >= 0.1 )
      {
        //log( "Выстрел firepower = " + firepower ) ;
        //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
        Bullet bullet = fireBullet( firepower ) ;
        if( bullet != null )
        {
          last_fire_time = cur_time ;
          if( cur_target != null )
            cur_target.shoot( firepower ) ;
        }
        //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
      }
      else
      {
        scan() ;
      }
    }

    // Вернуть направление пушки.
    public double
    cur_heading()
    { return cur_heading ; }
```

// Реализация автомата A1.

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

/* Метка 1. Протоколирование запуска автомата. Перейти на метку "2". */

```
      if( OBJECTS_LOGGING )
        log( "Для объекта 'Стрелок':" ) ;

      if( A1_BEGIN_LOGGING )
        log_begin( "A1", y1, e ) ;
```

/* Метка 2. Проверка номера состояния. Перейти на метку "3". */

```
      switch( y1 )
      {
        case 0:
          if(   (y2 == 1)
```

/* Метка 3. Проверка условия перехода. Перейти на метку "4". */

```
              && x25() )
```

```
/* Метка 5. Изменение переменной состояния. Перейти на метку "6". */
                                                        y1 = 3 ;
          else
          { z30() ; z70() ; z50_1() ; }
        break ;

        case 1:
          if( x26() )          { z80() ;                  y1 = 0 ; }
          else
          if( x30() && x22() )                            y1 = 2 ;
          else
          { z40() ; z50_0() ; }
        break ;

        case 2:
          if( x26() )          { z80() ;                  y1 = 0 ; }
          else
          if( x21() && x50() ) { z60() ;                  y1 = 0 ; }
          else
          if( !x30() )                                    y1 = 1 ;
          else
          { z40() ; z50_0() ; }
        break ;

        case 3:
          if( x26() )          { z80() ;                  y1 = 0 ; }
          else
          if( x20() )                                     y1 = 1 ;
          else
          { z50_1() ; }
        break ;

        default :
          if( A1_ERROR_LOGGING )
            log_error( "A1", y1 ) ;
      }

      if( y1 != y_old )
      {
/* Метка 6. Протоколирование перехода. Перейти на метку "7". */
        if( A1_TRANS_LOGGING )
          log_trans( "A1", y1, y_old ) ;

        switch( y1 )
        {
          case 0:
            z30() ; z70() ;
          break ;

          case 1:
            z40() ; z50_0() ;
          break ;

          case 2:
            z40() ; z50_0() ;
          break ;

          case 3:
/* Метка 7. Вызов фунции выходного воздействия. Перейти на метку "8". */
            z50_1() ;
          break ;
        }
        }

/* Метка 9. Протоколирование завершения работы автомата. Перейти на метку "10". */
        if( A1_END_LOGGING )
          log_end( "A1", y1 ) ;
      }

// Реализация автомата A2.
      private void
      A2( int e )
      {
        int   y_old = y2 ;

        if( A2_BEGIN_LOGGING )
```

```
        log_begin( "A2", y2, e ) ;

      switch( y2 )
      {
        case 0:
          if( x10() )  { z36() ;     y2 = 1 ; }
          else
          { z35() ; }
        break ;

        case 1:
        break ;

        default :
          if( A2_ERROR_LOGGING )
            log_error( "A2", y2 ) ;
      }

      if( y2 != y_old )
      {

      if( A2_TRANS_LOGGING )
        log_trans( "A2", y2, y_old ) ;
      }

      if( A2_END_LOGGING )
        log_end( "A2", y2 ) ;
    }


//*** Реализация входных переменных.
    private boolean
    x10()
    {
      boolean result = (cur_time > 3) && (gun_heat_decrement > 0) ;

      if( INPUTS_LOGGING )
        log_input( "x10", "Подсчет скорости охлаждения пушки завершен", result ) ;
      return result ;
    }

    private boolean
    x20()
    {
      boolean result = cur_gun_heat/gun_heat_decrement <= 3 ;

      if( INPUTS_LOGGING )
        log_input( "x20", "Пушка скоро охладится", result ) ;
      return result ;
    }

    private boolean
    x21()
    {
      boolean result = cur_gun_heat <= 0 ;

      if( INPUTS_LOGGING )
        log_input( "x21", "Пушка охладилась", result ) ;
      return result ;
    }

    private boolean
    x22()
    {
      boolean result = cur_gun_heat/gun_heat_decrement <= 1 ;

      if( INPUTS_LOGGING )
        log_input( "x22", "До конца охлаждения пушки меньше двух ходов", result ) ;
      return result ;
    }

    private boolean
    x25()
    {
/* Метка 4. Реализация входной переменной. */
      boolean result = cur_target != null ;
```

```
      if( INPUTS_LOGGING )
        log_input( "x25", "Цель выбрана", result ) ;
      return result ;
/* Перейти на метку "5". */
    }

    private boolean
    x26()
    {
      boolean result = true ;

      if( cur_target != null )
        result = !cur_target.is_tracked() ;

      if( INPUTS_LOGGING )
        log_input( "x26", "Цель потеряна", result ) ;
      return result ;
    }

    private boolean
    x30()
    {
      boolean result = true ;

      double  gun_to_go = get_angle_diff( cur_heading, cur_aim.a ) ;
      double  turn_direction = gun_to_go >= 0 ? 1 : -1 ;
                               // Обращение к объекту класса "Водитель".
      double  gun_turning_speed = turn_direction * gun_rotation_speed
                                + driver.turning_speed() ;

      result = Math.abs( gun_to_go / gun_turning_speed ) <= 1 ;

      if( INPUTS_LOGGING )
        log_input( "x30", "До конца поворота пушки меньше двух ходов", result ) ;
      return result ;
    }

    private boolean
    x50()
    {
      boolean result = true ;

      double  gun_to_go = get_angle_diff( cur_heading, cur_aim.a ) ;
                          //shortest_turn( cur_aim.a - cur_heading ) ;
      result = Math.abs(gun_to_go) < precision ;

      if( INPUTS_LOGGING )
        log_input( "x50", "Наводка правильная", result ) ;
      return result ;
    }

//*** Реализация выходных воздействий.
    private void
    z30()
    {
      if( OUTPUTS_LOGGING )
        log_output( "z30", "Выбрать цель" ) ;

      cur_target = targets.get_closest_target( 8 ) ;
      if( cur_target == null )
        cur_target = targets.closest_target ;
    }

    private void
    z35()
    {
      if( OUTPUTS_LOGGING )
        log_output( "z35", "Подсчет скорости охлаждения пушки" ) ;

      gun_heat_decrement = old_gun_heat - cur_gun_heat ;
    }

    private void
    z36()
    {
```

```
  if( OUTPUTS_LOGGING )
    log_output( "z36", "Вывести скорость охлаждения пушки" ) ;

  log( "gun_heat_decrement = " + gun_heat_decrement ) ;
}

private void
z40()
{
  if( OUTPUTS_LOGGING )
    log_output( "z40", "Рассчитать мощность выстрела" ) ;

  double  P = 0.25 ;

  if( cur_life >= life_ok )
  {
    P = 0.2 ;
    if( cur_robots_count > 2 )
    {
      P = 0.4 ;
    }
  }
  else
  if( cur_life <= life_warning )
  {
    P = 0.25 ;
    if( cur_robots_count > 2 )
    {
      P = 0.5 ;
    }
  }
  else
  if( cur_life <= life_critical )
  {
    P = 0.6 ;
  }

  cur_firepower = cur_target.get_firepower(P) ;
}

private void
z50_0()
{
  if( OUTPUTS_LOGGING )
    log_output( "z50_0", "Рассчитать точное упреждение и направить пушку" ) ;

  if( cur_target != null  )
  {
    vect_t predicted_pos ;

    predicted_pos = cur_target.predict_position_medium(
                      get_bullet_speed(cur_firepower), 2 ) ;

    cur_aim.reset( predicted_pos.a, predicted_pos.R ) ;
    da = get_angle_diff( cur_heading, predicted_pos.a ) ;
  }
}

private void
z50_1()
{
```

/* Метка 8. Реализация фунции выходного воздействия. */

```
  if( OUTPUTS_LOGGING )
    log_output( "z50_1",
                "Рассчитать приблизительное упреждение и направить пушку" ) ;

  if( cur_target != null  )
  {
    vect_t predicted_pos ;

    predicted_pos = cur_target.predict_position_roughly(
                      get_bullet_speed(base_firepower), 2 ) ;

    cur_aim.reset( predicted_pos.a, predicted_pos.R ) ;
    da = get_angle_diff( cur_heading, predicted_pos.a ) ;
  }
```

```
/* Перейти на метку "9". */
    }

    private void
    z60()
    {
      if( OUTPUTS_LOGGING )
        log_output( "z60", "Выстрел" ) ;

      firepower = cur_firepower ;

      //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
    }

    private void
    z70()
    {
      if( OUTPUTS_LOGGING )
        log_output( "z70", "Сбросить историю маневрирования цели" ) ;

      if( cur_target != null )
        cur_target.reset_speed_history() ;
    }

    private void
    z80()
    {
      if( OUTPUTS_LOGGING )
        log_output( "z80", "Сбросить текущую цель" ) ;

      cur_target = null ;
    }
  } // gunner_t
```

ПРИЛОЖЕНИЕ 2. РЕАЛИЗАЦИЯ КЛАССА "СТРЕЛОК" С ИСПОЛЬЗОВАНИЕМ ПАТТЕРНА "STATE"

```
//======
// Объектно-ориентированная реализация класса "Стрелок" с
// реализацией автоматов в других классах с применением
// шаблона "State". Реализация выполнена Д.В. Кузнецовым, СПбГИТМО (ТУ).
// Комментарии с символьными обозначениями переменных и компоновка текста:
// Шалыто А.А., Туккель Н.И.
// В тексте приведен путь от запуска автомата до завершения его работы.
// Этапы этого пути обозначены пронумерованными метками.
// В этом пути учтены не все выполняемые вызовы методов, например вызовы
// метода getA1state().
// Полученный путь включает 18 этапов.
// Глубина вложенности вызовов методов не менее четырех.
//

// Файл Gunner.java
package newCynic;

import robocode.Bullet;

//======
// Класс "Стрелок".
//

public class Gunner  {

//*** Атрибуты.

    // Направление пушки.
    private double _cur_heading;

    private double _old_gun_heat;

    // Температура пушки.
    private double _cur_gun_heat;

    // Скорость охлаждения пушки.
    private double gun_heat_decrement;
```

```
    // Текущая цель.
    private Target _cur_target;

    // Текущий прицел.
    private GeomVector _cur_aim;

    // Текущая мощность выстрела.
    private double _cur_firepower;

    // На сколько надо повернуть пушку на данном шаге.
    private double _da;

    // Мощность производимого выстрела.
    private double _firepower;

    // Ссылка на супервизора.
    private Cynical _robot;

    // Состояние автомата A1.
    private GunnerState _a1state;

    // Состояние автомата A2.
    private GunnerAuxState _a2state;

//*** Методы.

    // Конструктор.
    Gunner(Cynical aRobot) {
        _robot = aRobot;
        GunnerState.reset(this);
        GunnerAuxState.reset(this);
//        _A2_state = 0;
        gun_heat_decrement = 0;
        _cur_aim = new GeomVector();

    }

    // Начало раунда.
    public void begin_round() {
        _cur_heading = _robot.getGunHeadingRadians();
        _old_gun_heat = _robot.getGunHeat();
        _cur_gun_heat = _robot.getGunHeat();
        _cur_target = null;
        _cur_aim.reset();
        _robot._lastShotTime = 0;
    }

    // Начало шага.
    public void begin_turn() {
        _cur_heading = _robot.getGunHeadingRadians();
        _old_gun_heat = _cur_gun_heat;
        _cur_gun_heat = _robot.getGunHeat();

        _da = 0;
        _firepower = 0;

        // Запуск автоматов.
/* Метка 0. Запуск автомата A1. Перейти на метку "1". */
        GunnerState.processIncomingEvent(this, 10);
/* Метка 18. Завершение работы автомата A1. */
        GunnerAuxState.processIncomingEvent(this, 10);
    }

    // Конец шага.
    public void end_turn() {
        _robot.setTurnGunRightRadians(_da);

        if (_firepower >= 0.1) {
            Bullet bullet = _robot.fireBullet(_firepower);
            if (bullet != null) {
                _robot._lastShotTime = _robot._currentTime;
                if (_cur_target != null) {
                    _cur_target.shoot(_firepower);
                }
            }
        } else {
```

```
            _robot.scan();
        }
    }

    // Вернуть направление пушки.
    public double cur_heading() {
        return _cur_heading;
    }

    // Вернуть состояние автомата A1.
    public GunnerState getA1state() {
        return _a1state;
    }

    // Установить состояние автомата A1.
    public void setA1state(GunnerState aA1state) {
```

/* Метка 10. Изменение состояния. Перейти на метку "11". */

```
        _a1state = aA1state;
    }

    // Вернуть состояние автомата A2.
    public GunnerAuxState getA2state() {
        return _a2state;
    }

    // Установить состояние автомата A2.
    public void setA2state(GunnerAuxState aA2state) {
        _a2state = aA2state;
    }

//*** Реализация входных переменных.
    public boolean isGunCoolingRateCalculationFinished() {
        boolean result = (_robot._currentTime > 3) && (gun_heat_decrement > 0);

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x10",
                             "Подсчет скорости охлаждения пушки завершен", result);
        return result;
    }

    public boolean gunIsExpectedToBeCold() {
        boolean result = _cur_gun_heat / gun_heat_decrement <= 3;

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x20", "Пушка скоро охладится", result);
        return result;
    }

    public boolean gunIsCold() {
        boolean result = _cur_gun_heat <= 0;

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x21", "Пушка охладилась", result);
        return result;
    }

    public boolean gunWillBeColdWithinTwoSteps() {
        boolean result = _cur_gun_heat / gun_heat_decrement <= 1;

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x22", "До конца охлаждения пушки меньше двух ходов",
result);
        return result;
    }

    public boolean targetIsCaptured() {
```

/* Метка 6. Реализация входной переменной. */

```
        boolean result = _cur_target != null;

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x25", "Цель выбрана", result);
        return result;
```

/* Перейти на метку "7". */

```
    }
```

```
    public boolean targetIsLost() {
        boolean result = true;

        if (_cur_target != null)
            result = !_cur_target.isTracked();

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x26", "Цель потеряна", result);
        return result;
    }

    public boolean gunWillTurnWithinTwoSteps() {
        boolean result = true;
        double gun_to_go = Cynical.get_angle_diff(_cur_heading, _cur_aim.getAngle());
        double turn_direction = gun_to_go >= 0 ? 1 : -1;
        double gun_turning_speed = turn_direction * Cynical.MAX_GUN_ROTATION_SPEED
                + _robot.getDriver().turning_speed();

        result = Math.abs(gun_to_go / gun_turning_speed) <= 1;
        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x30", "До конца поворота пушки меньше двух ходов", result);
        return result;
    }

    public boolean isPointingFine() {
        boolean result = true;

        double gun_to_go = Cynical.get_angle_diff(_cur_heading, _cur_aim.getAngle());
//shortest_turn( cur_aim.a - cur_heading ) ;
        result = Math.abs(gun_to_go) < Cynical.PRECISION;

        if (Constants.INPUTS_LOGGING)
            Logger.log_input("x50", "Наводка правильная", result);
        return result;
    }

//*** Реализация выходных воздействий.
    public void selectTarget() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z30", "Выбрать цель");

        _cur_target = _robot.getTargets().get_closest_target(8);
        if (_cur_target == null)
            _cur_target = _robot.getTargets().closest_target;
    }

    void determineGunCoolingRate() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z35", "Подсчет скорости охлаждения пушки");

        gun_heat_decrement = _old_gun_heat - _cur_gun_heat;
    }

    public void logGunCoolingRate() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z36", "Вывести скорость охлаждения пушки");
        Logger.log("gun_heat_decrement = " + gun_heat_decrement);
    }

    public void calculateFirePower() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z40", "Рассчитать мощность выстрела");

        double P = 0.25;

        if (_robot._currentEnegy >= Cynical.ENERGY_NORMAL_THRESHOLD) {
            P = 0.2;
            if (_robot._aliveRobotsCount > 2) {
                P = 0.4;
            }
        } else if (_robot._currentEnegy <= Cynical.ENERGY_WARNING_THRESHOLD) {
            P = 0.25;
            if (_robot._aliveRobotsCount > 2) {
                P = 0.5;
            }
        } else if (_robot._currentEnegy <= Cynical.ENERGY_CRITICAL_THRESHOLD) {
            P = 0.6;
```

```
        }

        _cur_firepower = _cur_target.getOptimalFirePower(P);
    }

    public void calculateFineForestallingAndTurnGun() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z50_0", "Рассчитать точное упреждение и направить пушку");

        if (_cur_target != null) {
            GeomVector predicted_pos;

            predicted_pos = _cur_target.calculateAveragePrediction(
                    Cynical.get_bullet_speed(_cur_firepower), 2);

            _cur_aim.setCoords(predicted_pos.getAngle(), predicted_pos.getRadius());
            _da = Cynical.get_angle_diff(_cur_heading, predicted_pos.getAngle());
        }
    }

    public void calculateRoughForestallingAndTurnGun() {
/* Метка 13. Реализация выходного воздействия. */
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z50_1",
                              "Рассчитать приблизительное упреждение и направить пушку");

        if (_cur_target != null) {
            GeomVector predicted_pos;

            predicted_pos = _cur_target.calculateRoughPrediction(
                    Cynical.get_bullet_speed(Cynical.BASE_FIRE_POWER), 2);

            _cur_aim.setCoords(predicted_pos.getAngle(), predicted_pos.getRadius());
            _da = Cynical.get_angle_diff(_cur_heading, predicted_pos.getAngle());
        }
/* Перейти на метку 14. */
    }

    public void makeShot() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z60", "Выстрел");

        _firepower = _cur_firepower;

        //log( "Выстрел на gun = " + getGunHeadingRadians()*180/PI ) ;
    }

    public void dropTargetPathHistory() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z70", "Сбросить историю маневрирования цели");

        if (_cur_target != null)
            _cur_target.resetSpeedHistory();
    }

    public void dropCurrentTarget() {
        if (Constants.OUTPUTS_LOGGING)
            Logger.log_output("z80", "Сбросить текущую цель");
        _cur_target = null;
    }
} // gunner_t


// Классы, реализующие автомат A1.

// Файл GunnerState.java
package newCynic;

// Абстрактный класс "Состояние стрелка".
public abstract class GunnerState {

//*** Атрибуты.

    // Название состояния.
    private String _name;
```

```
    // Состояние 0 "Выбор цели".
    private static GunnerState STATE_0 = new A1STATE_0();

    // Состояние 1 "Сопровождение".
    private static GunnerState STATE_1 = new A1STATE_1();

    // Состояние 2 "Захват".
    private static GunnerState STATE_2 = new A1STATE_2();

    // Состояние 3 "Анализ цели".
    private static GunnerState STATE_3 = new A1STATE_3();

//*** Методы.

    // Конструктор.
    protected GunnerState(String aName) {
        _name = aName;
    }

    // Вернуть название состояния.
    public String getName() {
        return _name;
    }

    // Перевести автомат в начальное состояние.
    public static void reset(Gunner aGunMaster) {
        aGunMaster.setA1state(STATE_0);

    }

    // Переходы и действия на дугах и петлях.
    public abstract void processEvent(int aEvent, Gunner aGunMaster);

    // Изменение состояния автомата.
    protected void chageParentState(Gunner aGunMaster, GunnerState aNewState) {

/* Метка 8. Протоколирование перехода. Перейти на метку "9". */
        Logger.logStateChange(Constants.A1_AUTOMATE_NAME, aNewState.getName(),
                              aGunMaster.getA1state().getName());

/* Метка 9. Вызов функции, изменяющей состояние. Перейти на метку "10". */
        aGunMaster.setA1state(aNewState);

/* Метка 11. Вызов функции, формирующей выходные воздействия в новом состоянии.
Перейти на метку "12". */
        aNewState.onEnter(aGunMaster);

/* Метка 15. Завершение функции изменения состояния автомата. Перейти на метку "16". */
    }

    // Действия в состоянии.
    public abstract void onEnter(Gunner aGunMaster);

    // Запуск автомата.
    public static void processIncomingEvent(Gunner aGunMaster, int aEvent) {

/* Метка 1. Протоколирование запуска автомата. Перейти на метку "2". */
/* Текст функций протоколирования находится в другом файле и не приведен. */
        if (Constants.OBJECTS_LOGGING)
            Logger.log("Для объекта 'Стрелок':");

        if (Constants.A1_BEGIN_LOGGING)
            Logger.logBegin(Constants.A1_AUTOMATE_NAME,
              aGunMaster.getA1state().getName(), aEvent);

/* Метка 2. Собственно запуск автомата. Перейти на метку "3". */
        aGunMaster.getA1state().processEvent(aEvent, aGunMaster);

/* Метка 17. Протоколирование завершения работы автомата. Перейти на метку "18". */
        if (Constants.A1_END_LOGGING)
            Logger.logEnd(Constants.A1_AUTOMATE_NAME,
              aGunMaster.getA1state().getName());
    }

//*** Состояния.
```

```
    // Состояние 0 "Выбор цели".
    private static class A1STATE_0 extends GunnerState {

        // Конструктор.
        public A1STATE_0() {
            super("State 0");
        }

        // Переходы и действия на дугах и петлях.
        public void processEvent(int aEvent, Gunner aGunMaster) {

/* Метка 3. Проверка условия перехода. Перейти на метку "4". */
            if ((   aGunMaster.getA2state().isCalculationsFinished())  // y2 = 1
/* Метка 5. Проверка условия перехода. Перейти на метку "6". */
                 && aGunMaster.targetIsCaptured())                    // x25
            {
/* Метка 7. Начало действий при изменении состояния с "нулевого" на "третье". Перейти на метку "8". */
                chageParentState(aGunMaster, STATE_3);                // y1 = 3
/* Метка 16. Конец действий при изменении состояния. Перейти на метку "17". */
            }
            else
            {
                aGunMaster.selectTarget();                            // z30
                aGunMaster.dropTargetPathHistory();                   // z70
                aGunMaster.calculateRoughForestallingAndTurnGun();    // z50_1
            }
        }

        // Действия в состоянии.
        public void onEnter(Gunner aGunMaster) {
            aGunMaster.selectTarget();                                // z30
            aGunMaster.dropTargetPathHistory();                       // z70
        }
    }

    // Состояние 1 "Сопровождение".
    private static class A1STATE_1 extends GunnerState {

        // Конструктор.
        public A1STATE_1() {
            super("State 1");
        }

        // Переходы и действия на дугах и петлях.
        public void processEvent(int aEvent, Gunner aGunMaster) {
            if (aGunMaster.targetIsLost())                            // x26
            {
                aGunMaster.dropCurrentTarget();                       // z80
                chageParentState(aGunMaster, STATE_0);                // y1 = 0
            }
            else
            if (   aGunMaster.gunWillTurnWithinTwoSteps()             // x30
                && aGunMaster.gunWillBeColdWithinTwoSteps())          // x22
            {
                chageParentState(aGunMaster, STATE_2);                // y1 = 2
            }
            else
            {
                aGunMaster.calculateFirePower();                      // z40
                aGunMaster.calculateFineForestallingAndTurnGun();     // z50_0
            }
        }

        // Действия в состоянии.
        public void onEnter(Gunner aGunMaster) {
            aGunMaster.calculateFirePower();                          // z40
            aGunMaster.calculateFineForestallingAndTurnGun();         // z50_0
        }
    }

    // Состояние 2 "Захват".
    private static class A1STATE_2 extends GunnerState {

        // Конструктор.
        public A1STATE_2() {
```

```
            super("State 2");
        }

        // Переходы и действия на дугах и петлях.
        public void processEvent(int aEvent, Gunner aGunMaster) {
            if (aGunMaster.targetIsLost())                              // x26
            {
                aGunMaster.dropCurrentTarget();                         // z80
                chageParentState(aGunMaster, STATE_0);                  // y1 = 0
            }
            else
            if (   aGunMaster.gunIsCold()                               // x21
                && aGunMaster.isPointingFine())                         // x50
            {
                aGunMaster.makeShot();                                  // z60
                chageParentState(aGunMaster, STATE_0);                  // y1 = 0
            }
            else
            if (!aGunMaster.gunWillTurnWithinTwoSteps())                // !x30
            {
                chageParentState(aGunMaster, STATE_1);                  // y1 = 1
            }
            else
            {
                aGunMaster.calculateFirePower();                        // z40
                aGunMaster.calculateFineForestallingAndTurnGun();       // z50_0
            }
        }

        // Действия в состоянии.
        public void onEnter(Gunner aGunMaster) {
            aGunMaster.calculateFirePower();                            // z40
            aGunMaster.calculateFineForestallingAndTurnGun();           // z50_0
        }
    }

    // Состояние 3 "Анализ цели".
    private static class A1STATE_3 extends GunnerState {

        // Конструктор.
        public A1STATE_3() {
            super("State 3");
        }

        // Переходы и действия на дугах и петлях.
        public void processEvent(int aEvent, Gunner aGunMaster) {
            if (aGunMaster.targetIsLost())                              // x26
            {
                aGunMaster.dropCurrentTarget();                         // z80
                chageParentState(aGunMaster, STATE_0);                  // y1 = 0
            }
            else
            if (aGunMaster.gunIsExpectedToBeCold())                     // x20
            {
                chageParentState(aGunMaster, STATE_1);                  // y1 = 1
            }
            else
            {
                aGunMaster.calculateRoughForestallingAndTurnGun();      // z50_1
            }
        }

        // Действия в состоянии.
        public void onEnter(Gunner aGunMaster) {

/* Метка 12. Вызвать функцию выходного воздействия. Перейти на метку "13". */
            aGunMaster.calculateRoughForestallingAndTurnGun();          // z50_1
/* Метка 14. Завершение функции, вызывающей функции выходных воздействий.
Перейти на метку "15". */
        }
    }
}
```

```
// Классы, реализующие автомат A2

// Файл GunnerAuxState.java
package newCynic;

// Абстрактный класс "Состояние автомата подсчета скорости охлаждения пушки".
public abstract class GunnerAuxState {

//*** Атрибуты.

    // Название состояния.
    private String _name;

    // Состояние 0 "Подсчет".
    private final static GunnerAuxState STATE_0 = new A2STATE_0();

    // Состояние 1 "Подсчет окончен".
    private final static GunnerAuxState STATE_1 = new A2STATE_1();

//*** Методы.

    // Конструктор.
    protected GunnerAuxState(String aName) {
        _name = aName;
    }

    // Вернуть название состояния.
    public String getName() {
        return _name;
    }

    // Сообщить, окончен ли подсчет скорости охлаждения.
    public boolean isCalculationsFinished() {
        return false;
    }

    // Перевести автомат в начальное состояние.
    public static void reset(Gunner aGunMaster) {
        aGunMaster.setA2state(STATE_0);
    }

    // Переходы и действия на дугах и петлях.
    public abstract void processEvent(int aEvent, Gunner aGunMaster);

    // Изменение состояния автомата.
    protected void chageParentState(Gunner aGunMaster, GunnerAuxState aNewState) {
        Logger.logStateChange(Constants.A2_AUTOMATE_NAME,
          aNewState.getName(), aGunMaster.getA2state().getName());
        aGunMaster.setA2state(aNewState);
        aNewState.onEnter(aGunMaster);
    }

    // Действия в состоянии.
    public abstract void onEnter(Gunner aGunMaster);

    // Запуск автомата.
    public static void processIncomingEvent(Gunner aGunMaster, int aEvent) {
        if (Constants.A2_BEGIN_LOGGING)
           Logger.logBegin(Constants.A2_AUTOMATE_NAME,
             aGunMaster.getA2state().getName(), aEvent);

        aGunMaster.getA2state().processEvent(aEvent, aGunMaster);

        if (Constants.A2_END_LOGGING)
            Logger.logEnd(Constants.A2_AUTOMATE_NAME, aGunMaster.getA2state().getName());
    }

//*** Состояния.

    // Состояние 0 "Подсчет".
    private static class A2STATE_0 extends GunnerAuxState {

        // Конструктор.
        public A2STATE_0() {
            super("State 0");
        }
```

```
        // Переходы и действия на дугах и петлях.
        public void processEvent(int aEvent, Gunner aGunMaster) {
            if (aGunMaster.isGunCoolingRateCalculationFinished())    // x10
            {
                aGunMaster.logGunCoolingRate();                      // z36
                chageParentState(aGunMaster, STATE_1);               // y2 = 1
            } else {
                aGunMaster.determineGunCoolingRate();                // z35
            }
        }

        // Действия в состоянии.
        public void onEnter(Gunner aGunMaster) {
        }
    }

    // Состояние 1 "Подсчет окончен".
    private static class A2STATE_1 extends GunnerAuxState {

        // Конструктор.
        public A2STATE_1() {
            super("State 1");
        }

        // Переходы и действия на дугах и петлях.
        public void processEvent(int aEvent, Gunner aGunMaster) {
        }

        // Действия в состоянии.
        public void onEnter(Gunner aGunMaster) {
        }

        // Сообщить, окончен ли подсчет скорости охлаждения.
        public boolean isCalculationsFinished() {
/* Метка 4. Проверить, окончен ли подсчет. Перейти на метку "5".*/
            return true;
        }
    }
}
```

ОБ АВТОРАХ

Шалыто Анатолий Абрамович — ученый секретарь Федерального научно-производственного центра (ФНПЦ) — Федерального государственного унитарного предприятия (ФГУП) "НПО "Аврора"", профессор кафедры "Компьютерные технологии" Санкт-Петербургского государственного института точной механики и оптики (технического университета) (СПбГИТМО (ТУ)).

С ним можно связаться по адресу: mail@avrorasystems.com ("для Шалыто").

Туккель Никита Иосифович — инженер-программист ФНПЦ — ФГУП "НПО "Аврора"".

С ним можно связаться по адресу: cynical@mail.ru.