Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

П.И. Фельдман, А.А. Шалыто

Объектно-ориентированная модификация автоматного подхода (на примере системы анимации моделей)

Объектно-ориентированное программирование с явным выделением состояний

Проектная документация

Проект создан в рамках
“Движения за открытую проектную документацию”
http://is.ifmo.ru

Санкт-Петербург

2004

Содержание

Содержание 2
Введение 4
1. Постановка задачи 5
2. Описание библиотеки «Auto-Lib» 5
2.1. Назначение библиотеки 5
2.2. Класс «BaseState» 6
2.2.1. Словесное описание 6
2.2.2. Краткое описание методов 6
2.3. Класс «BaseEvent» 6
2.3.1. Словесное описание 6
2.3.2. Краткое описание методов 6
2.4. Класс «BaseParam» 6
2.4.1. Словесное описание 6
2.4.2. Краткое описание методов 6
2.5. Класс «BaseAction» 7
2.5.2. Краткое описание методов 7
2.6. Класс «BaseAutomat» 7
2.6.1. Словесное описание 7
2.6.2. Краткое описание методов 7
2.7. Класс «DescriptedState» 8
2.7.1. Словесное описание 8
2.7.2. Краткое описание методов 8
2.8. Класс «DescriptedEvent» 9
2.8.1. Словесное описание 9
2.8.2. Краткое описание методов 9
2.9. Класс «DescriptedParam» 9
2.9.1. Словесное описание 9
2.9.2. Краткое описание методов 9
2.10. Класс «DescriptedAction» 9
2.10.1. Словесное описание 9
2.10.2. Краткое описание методов 9
2.11. Класс «DescriptedAutomat» 10
2.11.1. Словесное описание 10
2.11.2. Краткое описание методов 10
2.12. Класс «BaseLogger» 10
2.12.1. Словесное описание 10
2.12.2. Краткое описание методов 10
2.13. Класс «LoggedAutomat» 11
2.13.1. Словесное описание 11
2.13.2. Краткое описание методов 12
2.14. Работа с библиотекой «Auto-Lib» 12
3. Пример реализации анимации 13
3.1. Описание функций и структур, использованных для анимации 13
3.2. Класс-автомат A0 («Модель») 14
3.2.1. Словесное описание 14
3.2.2. Перечень состояний 14
3.2.3. Перечень событий 15
3.2.4. Перечень входных переменных 15
3.2.5. Перечень выходных воздействий 15
3.2.6. Схема связей 16
3.2.7. Граф переходов 17

3.3. Класс-автомат A1 («Загрузчик») 17
3.3.1. Словесное описание 17
3.3.2. Перечень состояний 18
3.3.3. Перечень событий 18
3.3.4. Перечень входных переменных 18
3.3.5. Перечень выходных воздействий 18
3.3.6. Схема связей 18
3.3.7. Граф переходов 19
3.4. Класс-автомат A2 («Часть») 19
3.4.1. Словесное описание 19
3.4.2. Перечень состояний 19
3.4.3. Перечень событий 20
3.4.4. Перечень входных переменных 20
3.4.5. Перечень выходных воздействий 20
3.4.6. Схема связей 20
3.4.7. Граф переходов 21
3.5. Класс «FileLogger» 22
3.5.1. Словесное описание 22
3.6. Диаграмма классов 23
3.7. Схема взаимодействия автоматов 24
Заключение 24
Источники 25
Приложение. Часть исходного кода 26
Библиотека «Auto-Lib» 26
Реализация автомата «Модель» (A0) 35
Реализация автомата «Загрузчик» (A1) 36
Реализация автомата «Часть» (A2) 36
Пример фрагмента протокола 38

Введение

В настоящее время существует несколько различных технологий, позволяющих описывать и создавать конечные автоматы, в частности, SWITCH-технология [1], с которой можно ознакомиться на сайтах http://is.ifmo.ru и http://www.softcraft.ru. Отличительной особенностью технологии является изоморфность графа переходов, входящего в проектную документацию, с кодом, реализующим конечный автомат в программе.

В данной работе эта технология была взята за основу и доработана с целью дальнейшего развития совместного использования автоматного и объектно-ориентированного подходов [2]. Разработан набор классов, объединенных в библиотеку «Auto-Lib», который позволяет описывать автоматы по «классической» SWITCH-технологии. Он обеспечивает возможность «прозрачного протоколирования» - в основном блоке автомата нет ни строчки кода, указывающей на протоколирование. Работу по протоколированию берет на себя указанный набор классов, что позволяет повысить централизацию логики работы системы и отделить ее от конкретной реализации. Кроме того, этот набор позволяет применять такой гибкий способ повторного использования кода, как наследование автоматов, что дает возможность переопределять и доопределять реакции автоматов на события.

Этот подход был успешно применен при реализации анимированных моделей, визуализатор одной из которых которой приведен на рис. 1.

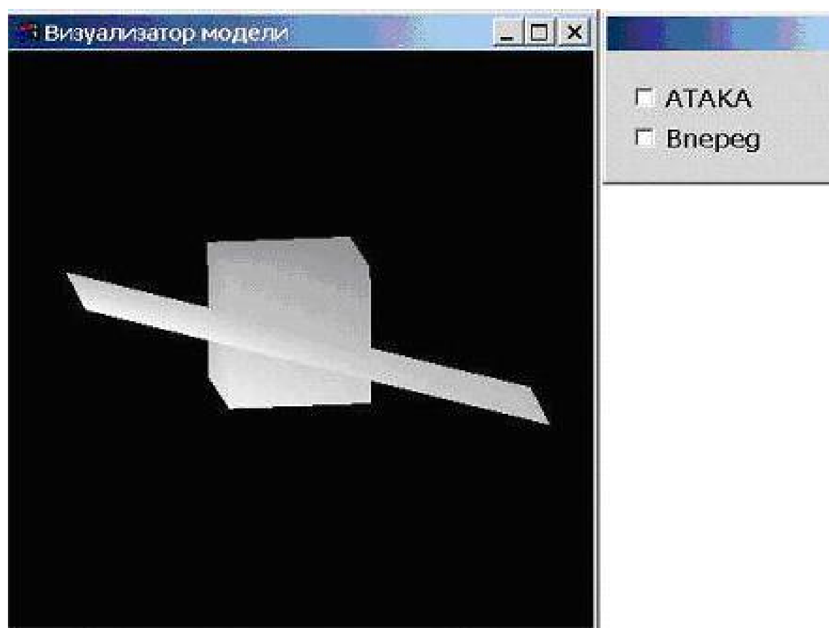


Рис. 1. Визуализатор модели

В заключение раздела отметим, что в настоящее время подходы, аналогичные предлагаемому, рассматриваются и в других работах [3, 4]. При этом описываемый подход отличается от рассмотренного в работе [4], в частности, тем, что в нем понятие «события» используется в явном виде, что более естественно для построения событийных систем. Этот подход позволяет не только «очистить» собственно автоматы от функций протоколирования, но и передавать в событиях дополнительную информацию для объединения событий, обладающих одинаковыми свойствами. Он обеспечивает возможность хранения дополнительной информации в состояниях для объединения однотипных состояний.

1. Постановка задачи

Требуется разработать набор инструментов (библиотеку), позволяющих быстро создавать и отображать анимированные модели. В качестве редактора их статических кадров (простых 3-х мерных фигур) используется программный продукт «3D Studio» фирмы *Kinetix*.

Суть отображения указанных моделей иллюстрируется примером.

Допустим, что модель реализует внешний вид стрелка, который выполняет один или несколько выстрелов. Выстрелы выполняются следующим образом: стрелок достает оружие, совершает один или несколько выстрелов и убирает его. Эти три этапа были названы предварительным, циклическим и завершающим этапами соответственно. При этом, если модель начала функционировать, то:

- выполнится предварительный этап (стрелок достанет оружие);
- как минимум один раз выполнится циклический этап (стрелок выстрелит);
- при необходимости продолжения стрельбы циклический этап повторяется (стрелок продолжит стрелять);
- если не следует совершить более приоритетное действие, то выполнится завершающий этап (оружие будет убрано);
- на завершающем этапе при необходимости выполнить более приоритетное действие, чем совершаемое на данные момент, переход в другой анимационный ряд должен произойти, как только возникла необходимость в этом переходе.
- при отсутствии циклического этапа, модель по окончании предварительного этапа сразу переходит в завершающий. После окончания завершающего этапа модель переходит в начало наиболее приоритетного анимационного ряда.

Модель состоит из частей, для каждой из которых определены анимационные ряды для тех видов поведения, которые она реализует.

Описанный способ отображения является модификацией предложенного в статье [5].

Преложим библиотеку, которая позволяет эффективно реализовать описанное выше поведение модели. Такую библиотеку целесообразно строить, используя не «классическое» объектно-ориентированное программирование, а объектно-ориентированное программирование с явным выделением состояний [6].

2. Описание библиотеки «Auto-Lib»

2.1. Назначение библиотеки

Библиотека «Auto-Lib» содержит набор классов, позволяющих создавать «прозрачно протоколируемые» автоматы, а также легко реализовывать автоматы, как по «классической» SWITCH-технологии, так и по ее модификации. При этом под модификацией этой технологии будем понимать возможность передачи в качестве события не только его номера, но и некоторой дополнительной информации.

Для этого предусмотрен вызов автомата, как с *событием-числом*, так и с *событием-объектом*. При использовании событий-объектов рекомендуется получать дополнительную информацию из события не в блоке реализации автомата, а в блоках реализации входных переменных и выходных воздействий. Это делает «объектно-событийный» автомат более похожим на «число-событийный» автомат, являющийся основой SWITCH-технологии.

Кроме того, предусмотрено применение *состояний-объектов*, доступ к полям которых также целесообразно производить из блоков реализации входных переменных и выходных воздействий.

Обратим внимание, что в дальнейшем используются такие термины, как «событие с описанием», «состояние с описанием», «входная переменная с описанием», «выходное воздействие с описанием» и «автомат с описанием». Описания этих сущностей применяются для протоколирования. Для задания сущностей с описаниями предусмотрена регистрация, суть которой состоит в перечислении сущностей и их описаний. В рамках предлагаемого подхода взаимодействие автоматов сведено к взаимодействию объектов.

2.2. Класс «BaseState»

2.2.1. Словесное описание

Реализует состояние. Содержит идентификатор состояния.

2.2.2. Краткое описание методов

`BaseState(uint32 y)` – конструктор, который задает идентификатор состояния.

2.3. Класс «BaseEvent»

2.3.1. Словесное описание

Реализует событие. Содержит идентификатор события.

2.3.2. Краткое описание методов

`BaseEvent(uint32 e)` – конструктор, который задает идентификатор события.

2.4. Класс «BaseParam»

2.4.1. Словесное описание

Предоставляет информацию о входной переменной. Содержит идентификатор входной переменной.

2.4.2. Краткое описание методов

`BaseParam(uint32 x)` – конструктор, который задает идентификатор входной переменной.

2.5. Класс «BaseAction»

2.5.1. Словесное описание

Предоставляет информацию о выходном воздействии. Содержит идентификатор выходного воздействия.

2.5.2. Краткое описание методов

`BaseAction(uint32 z)` – конструктор, который задает идентификатор выходного воздействия.

2.6. Класс «BaseAutomat»

2.6.1. Словесное описание

Этот класс реализует базовую функциональность автомата. Он содержит идентификатор автомата, номер текущего состояния, указатель на текущее состояние-объект, номер последнего вызванного события, указатель на последнее вызванное событие-объект, а также списки зарегистрированных автоматом событий, состояний, входных переменных и выходных воздействий. Предоставляет методы для регистрации состояний, событий, входных переменных и выходных воздействий. Позволяет своим потомкам переопределять методы реализации автомата, вычислений входных переменных и вызовов выходных воздействий. Предоставляет также методы для вызова автомата, как с событием-числом, так и с событием-объектом.

2.6.2. Краткое описание методов

`BaseAutomat(uint32 a)` – конструктор, который задает идентификатор автомата и инициализирует внутренние данные;

`virtual ~BaseAutomat()` – деструктор, который высвобождает занятые ресурсы;

`uint32 Get_y()` – возвращает идентификатор текущего состояния;

`virtual BaseState *RegisterState(uint32 y)` – регистрирует состояние с заданным идентификатором;

`void AddState(BaseState *Y)` – добавляет состояние в список состояний;

`virtual BaseEvent *RegisterEvent(uint32 e)` – регистрирует событие с заданным идентификатором;

`void AddEvent(BaseEvent *E)` – добавляет событие в список событий;

`virtual BaseParam *RegisterParam(uint32 x)` – регистрирует входную переменную с заданным идентификатором;

`void AddParam(BaseParam *X)` – добавляет входную переменную в список входных переменных;

`virtual BaseAction *RegisterAction(uint32 z)` – регистрирует выходное воздействие с заданным идентификатором;

`void AddAction(BaseAction *Z)` – добавляет выходное воздействие в список выходных воздействий;

`virtual void Main()` – реализация автомата (в данном классе пустая, так как предполагает переопределение в потомках);

`void BaseRun(BaseEvent *E)` – неперегружаемая функция вызова автомата с событием-объектом для внутреннего использования;

`virtual void Run(BaseEvent *E)` – перегружаемая функция вызова автомата с событием-объектом для внешнего использования;

`void virtual run(uint32 e)` – вызов автомата с событием-числом;

`virtual bool x(uint32 n)` – запуск вычисления n-ой входной переменной. Вызывается из метода реализации автомата;

`virtual bool Param(uint32 n)` – реализация вычисления n-ой входной переменной (предполагает перегрузку в потомках);

`virtual void z(uint32 n)` – запуск n-го выходного воздействия. Вызывается из метода реализации автомата;

`virtual void Action(uint32 n)` – реализация n-го выходного воздействия (предполагает перегрузку в потомках).

2.7. Класс «DescriptedState»

2.7.1. Словесное описание

Реализует состояние с описанием. Дополняет класс «BaseState» описанием.

2.7.2. Краткое описание методов

`DescriptedState(uint32 y,xstring szStateDescription)` – конструктор, который задает идентификатор и описание;

`DescriptedState(DescriptedState &Y)` – конструктор копирования, который создает копию переданного объекта.

2.8. Класс «DescriptedEvent»

2.8.1. Словесное описание

Реализует событие с описанием. Дополняет класс «BaseEvent» описанием.

2.8.2. Краткое описание методов

`DescriptedEvent(uint32 e,xstring szStateDescription)` – конструктор, который задает идентификатор и описание.

2.9. Класс «DescriptedParam»

2.9.1. Словесное описание

Содержит данные о входной переменной с описанием. Дополняет класс «BaseParam» описанием.

2.9.2. Краткое описание методов

`DescriptedParam(uint32 x,xstring szStateDescription)` – конструктор, который задает идентификатор и описание.

2.10. Класс «DescriptedAction»

2.10.1. Словесное описание

Реализует выходное воздействие с описанием. Дополняет класс «BaseAction» описанием.

2.10.2. Краткое описание методов

`DescriptedAction(uint32 z,xstring szStateDescription)` – конструктор, который задает идентификатор и описание.

2.11. Класс «DescriptedAutomat»

2.11.1. Словесное описание

Реализует автомат с описанием. Дополняет класс «BaseAutomat» описанием, а также методами для регистрации событий, состояний, входных переменных и выходных воздействий с соответствующими описаниями.

2.11.2. Краткое описание методов

`DescriptedAutomat(uint32 a, xstring szAutomatDescription)` – конструктор, который задает идентификатор и описание;

`virtual ~DescriptedAutomat()` – деструктор, который реализован пустым и предполагает переопределение;

`virtual DescriptedState *RegisterState(uint32 y,xstring szStateDescription)` – регистрирует состояние с описанием;

`virtual DescriptedEvent *RegisterEvent(uint32 e, xstring szEventDescription)` – регистрирует событие с описанием;

`virtual DescriptedParam *RegisterParam(uint32 x, xstring szParamDescription)` – регистрирует входную переменную с описанием;

`virtual DescriptedAction *RegisterAction(uint32 z, xstring szActionDescription)` – регистрирует выходное воздействие с описанием;

2.12. Класс «BaseLogger»

2.12.1. Словесное описание

Обеспечивает ведение протокола. Предоставляет набор методов для занесения в протокол практически любого действия с автоматом. Все методы реализованы пустыми, так как данный класс выполняет функции интерфейса. Однако сам он является классом для того, чтобы освободить программиста от необходимости формально переопределять методы, в реализации которых нет необходимости.

2.12.2. Краткое описание методов

`BaseLogger()` – конструктор;

`virtual ~BaseLogger()` – деструктор;

`virtual LogAutomatCreated(DescriptedAutomat *A)` – протоколирует создание автомата;

`virtual LogAutomatFinished(DescriptedAutomat *A)` – протоколирует завершение работы автомата;

`virtual LogStateRegistered(DescriptedAutomat *A,DescriptedState *Y) –` протоколирует регистрацию состояния;

`virtual LogEventRegistered(DescriptedAutomat *A,DescriptedEvent *Y) –` протоколирует регистрацию события;

`virtual LogParamRegistered(DescriptedAutomat *A,DescriptedParam *X) –` протоколирует регистрацию входной переменной;

`virtual LogActionRegistered(DescriptedAutomat *A,DescriptedAction *Z) –` протоколирует регистрацию выходного воздействия;

`virtual LogStateChanged(DescriptedAutomat *A, DescriptedState *Old_Y,DescriptedState *Y) –` протоколирует изменение состояния;

`virtual LogEventRaised(DescriptedAutomat *A,DescriptedEvent *E) –` протоколирует начало обработки события;

`virtual LogEventFinished(DescriptedAutomat *A,DescriptedEvent *E) –` протоколирует окончание обработки события;

`virtual LogParamCalcStarted(DescriptedAutomat *A, DescriptedParam *X) –` протоколирует начало вычисления входной переменной;

`virtual LogParamCalcFinished(DescriptedAutomat *A, DescriptedParam *X, bool x) –` протоколирует окончание вычисления входной переменной;

`virtual LogActionStarted(DescriptedAutomat *A, DescriptedAction *Z) –` протоколирует начало вызова выходного воздействия;

`virtual LogActionFinished(DescriptedAutomat *A, DescriptedAction *Z) –` протоколирует окончание вызова выходного воздействия;

`virtual LogString(xstring szComment) –` протоколирует произвольную строку;

2.13. Класс «LoggedAutomat»

2.13.1. Словесное описание

Реализует протоколируемый автомат. Переопределяет методы класса «DescriptedAutomat» так, чтобы каждое из перечисленных при описании класса «BaseLogger» действий заносилось в протокол.

2.13.2. Краткое описание методов

`LoggedAutomat(uint32 a,xstring szAutomatDescription,BaseLogger *Logger)` – конструктор, который задает идентификатор, описание и «составителя» протокола, а также протоколирует завершение создания автомата;

`~LoggedAutomat()` – деструктор, который протоколирует завершение работы автомата;

`virtual DescriptedState* RegisterState(uint32 y,xstring szStateDescription)` – регистрирует состояние с описанием и протоколирует факт регистрации;

`virtual DescriptedEvent* RegisterEvent(uint32 e,xstring szEventDescription)` – регистрирует событие с описанием и протоколирует факт регистрации;

`virtual DescriptedParam* RegisterParam(uint32 x,xstring szParamDescription)` – регистрирует входную переменную с описанием и протоколирует факт регистрации;

`virtual DescriptedAction* RegisterAction(uint32 y,xstring szActionDescription)` – регистрирует выходное воздействие с описанием и протоколирует факт регистрации;

`void virtual Run(BaseEvent *E)` – вызов автомата с событием-объектом, протоколирование вызова события, смены состояния и конца обработки события;

`virtual bool x(uint32 n)` – запуск вычисления n-ой входной переменной, протоколирование начала и конца вычисления;

`virtual void z(uint32 n)` – запуск выходного воздействия, протоколирование начала и конца выполнения выходного воздействия.

2.14. Работа с библиотекой «Auto-Lib»

На основе класса «LoggedAutomat», «завершающего» иерархию автоматных классов рассматриваемой библиотеки, можно создавать автоматы для прикладной задачи, регистрируя в конструкторе необходимые состояния, события, входные переменные и выходные воздействия, устанавливая для них индентификаторы и описания. Для реализации конкретного автомата следует переопределить метод `void Main()`, в котором должна быть реализована все поведение автомата.

Событие-число, с которым вызван автомат, хранится в поле `e`. Указатель на событие-объект, с которым вызван автомат, хранится в поле `E`, которое является указателем на класс «BaseEvent». В случае необходимости обработки события-объекта, в обработчике следует привести указатель `E` к соответствующему типу и продолжить работу с полученным указателем.

Состояние-число, в котором находится автомат, хранится в поле `y`. Указатель на состояние-объект, в котором находится автомат, хранится в поле `Y`, которое является указателем на класс «BaseState». В случае необходимости использования состояния-объекта, в обработчике следует привести указатель `Y` к соответствующему типу и продолжить работу с полученным указателем.

Для вызова автомата с событием-числом следует использовать метод `void run(uint32 e)`, передав ему в параметре номер зарегистрированного события. В случае отсутствия зарегистрированного события с таким номером вызовется исключение. Для вызова

автомата с событием-объектом следует применять метод `void Run(BaseEvent *E)`, передав ему в параметре указатель на соответствующий экземпляр класса, реализующего событие.

При изменении состояния-числа следует обращаться к полю `y`, присваивая ему значение номера зарегистрированного состояния. В случае отсутствия зарегистрированного состояния с номером, присвоенным полю `y`, вызовется исключение.

Для реализации вычисления входной переменной следует переопределить метод `bool Param(uint32 n)`, где `n` – номер входной переменной. Для вычисления входной переменной из метода реализации автомата следует вызывать метод `bool x(uint32 n)`, передав ему номер входной переменной.

Для реализации выходного воздействия следует переопределить метод `void Action(uint32 n)`, где `n` – номер выходного воздействия. Для вызова выходного воздействия из метода реализации автомата следует вызывать метод `void z(uint32 n)`, передав ему номер выходного воздействия.

3. Пример реализации анимации

3.1. Описание функций и структур, использованных для анимации

Перечислим функции, используемые для загрузки модели.

Функция `uint32 get_uint32(uint **parr)` - заружает беззнаковое 32-х битное число из блока памяти;

Функция `char get_char(uint **parr)` - загружает символ из блока памяти;

Функция `flt get_flt(uint **parr)` - загружает число с плавающей точкой из блока памяти;

Функция `char* get_str(uint **parr)` - загружает строку из блока памяти;

Перечислим структуры для хранения модели.

Структура `TmVertex` - содержит пять координат точки – три пространственные и две текстурные. В данном примере текстурные координаты не используются;

Структура `TmFrame` - содержит набор точек, экземпляров класса TmVertex;

Структура `TmFace` - содержит три индекса точек – вершин грани;

Структура `TmGroup` - содержит данные, описывающие анимационную группу:

- идентификатор действия, реализуемого данной группой;
- приоритет группы;
- количество кадров в предварительном, циклическом и завершающем этапах;
- время, необходимое для прохождения предварительного, циклического и завершающего этапов;
- время для совместного прохождения предварительного и циклического этапов;
- время для прохождения всей анимационной группы;
- набор указателей на кадры данной группы;

- набор величин, указывающих на то, сколько времени требуется для перехода в каждый кадр.

Структура `TmPart` – содержит данные, описывающие соответствующую часть модели:
- количество точек;
- количество граней;
- набор точек;
- набор граней;
- количество групп;
- набор групп;
- отдельно выделенную группу «бездействие»;
- указатель на текущую группу;
- указатель на следующую группу;
- положение текущего кадра в группе;
- указатель на предыдущий кадр;
- индекс следующего кадра;
- время прошедшее после предыдущего кадра.

3.2. Класс-автомат A0 («Модель»)

3.2.1. Словесное описание

Реализует поведение модели в целом. Содержит все данные о модели: ее имя, количество и набор частей, количество и набор точек в ограничивающем модель многограннике, количество и набор граней ограничивающего модель многогранника, версию модели, индекс текстуры (не используется в примере), время последнего вызова автомата и разницу во времени между последним вызовом автомата и текущим. Содержит в себе автомат A1 («Загрузчик») и набор автоматов A2 («Часть»), а также методы для их создания, уничтожения и работы с ними. Реализован, как потомок класса «LoggedAutomat». Получает как события-числа, так и события-объекты. Состояния являются числами. Всего состояний 2^{15}+1. Ввиду огромного количества состояний, автомат реализован не по SWITCH-технологии. Для иллюстрации его работы на рис. 3 приведен граф переходов, содержащий не все состояния, а только те из них, которые используются визуализирующей системой.

3.2.2. Перечень состояний

В визуализирующей системе будут применяться состояния только с номерами 300000, 0, 1, 64, 65, отвечающие за «не готовность», «готовность-бездействие», «готовность-атаку», «готовность-движение», «готовность-атаку-движение».

Приведем список номеров состояний:
300000 – «Не готов к работе» (начальное состояние);
0 – «Готов-бездействие»;
еще 2^{15}-1 состояний готовности, отличающиеся набором активированных действий.

3.2.3. Перечень событий

6 – подготовиться к работе;
7 – завершить работу;
8 – событие-объект вида «Прошло *time* времени, активировано действие *action*» либо «Прошло *time времени*, деактивировано действие *action*», которое условно обозначается «Прошло время».

Действие *action* может означать: «атака-1», «атака-2», «движение налево», «движение направо», «движение вперед», «движение назад», «движение вверх», «движение вниз», «поворот налево», «поворот направо», «поворот вверх», «поворот вниз», «поворот по часовой стрелке», «поворот против часовой стрелки», «боль». В визуализирующей системе в качестве действия *action* будут использованы только такие действия, как «атака-1» и «движение вперед».

3.2.4. Перечень входных переменных

0 – Действие активировано;
1 – Действие - атака-1;
2 – Действие - атака-2;
3 – Действие - движение влево;
4 – Действие - движение вправо;
5 – Действие - движение вверх;
6 – Действие - движение вниз;
7 – Действие - движение вперед;
8 – Действие - движение назад;
9 – Действие - поворот налево;
10 – Действие - поворот направо;
11 – Действие - поворот вверх;
12 – Действие - поворот вниз;
13 – Действие - поворот по часовой стрелке;
14 – Действие - поворот против часовой стрелки;
15 – Действие – боль.

Смысл применяемых обозначений состоит в следующем: входная переменная `x` устанавливается в значение `true`, если было активировано соответствующее действие.

В визуализирующей системе используются только входные переменные с номерами ноль, один, семь.

3.2.5. Перечень выходных воздействий

0 – Создать «Загрузчик»
1 – Посредством «Загрузчика» загрузить модель, проверить корректность
2 – Создать части
3 – Завершить работу частей
4 – Посредством «Загрузчика» выгрузить модель
5 – Завершить работу «Загрузчика»
6 – Запомнить время вызова
7 – Установить смещение по времени частям

3.2.6. Схема связей

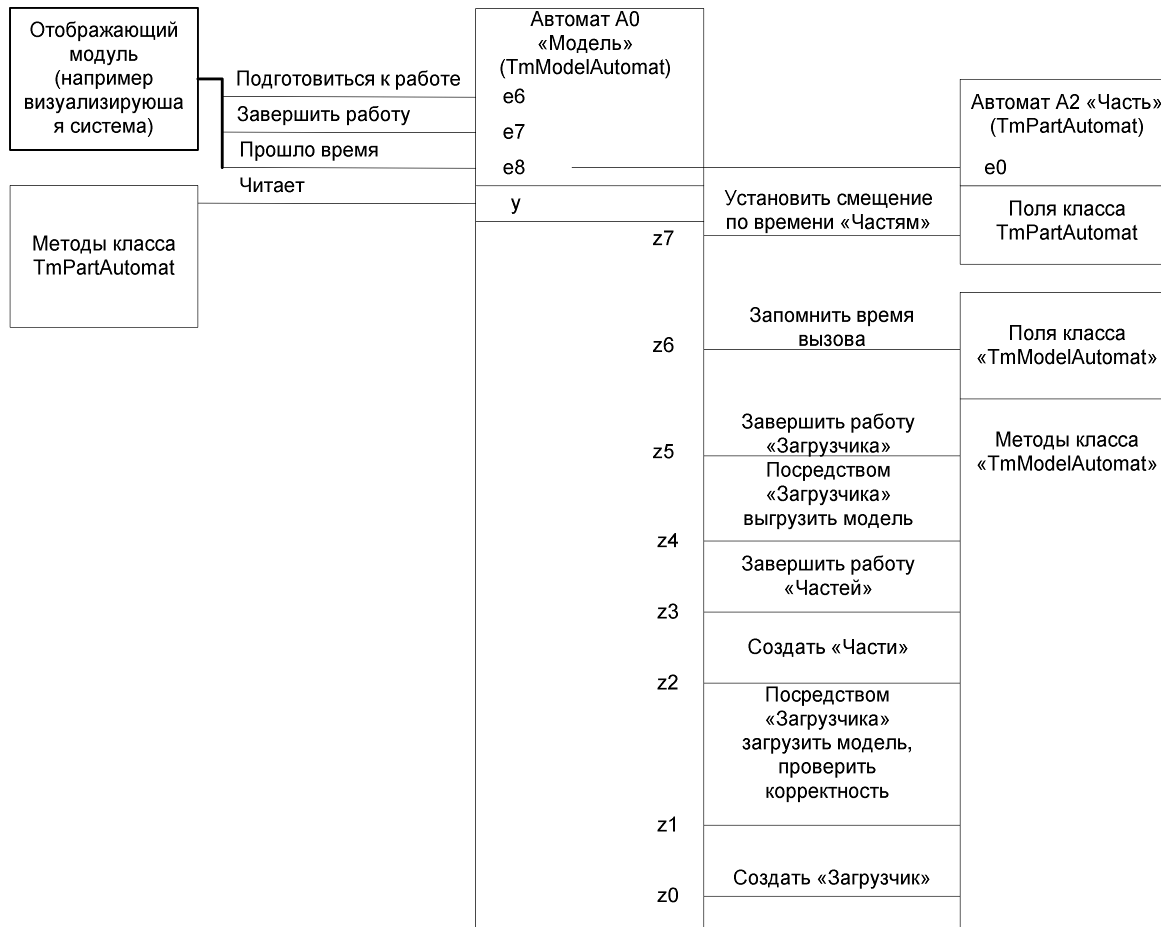
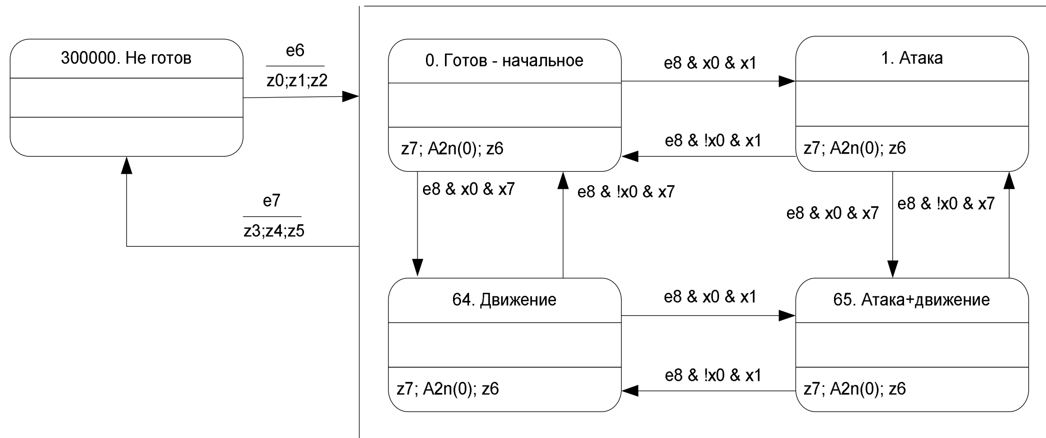


Рис. 2. Схема связей автомата «Модель» (А0)

Обратим внимание, что в этой схеме отображено, что событие e8, являющееся входным для автомата А0, инициирует событие e0 - входное для автомата А2.

3.2.7. Граф переходов

Из-за большого количества состояний представить полный граф переходов невозможно. Далее приведен упрощенный граф переходов, с 5-ю состояниями, которые используются в визуализирующей системе.



x0	Действие активировано
x1	Действие - атака-1
x7	Действие - движение вперед

e6	Подготовиться к работе
e7	Завершить работу
e8	Прошло время

z0	Создать «Загрузчик»
z1	Посредством «Загрузчика» загрузить модель, проверить корректность
z2	Создать «Части»
z3	Завершить работу «Частей»
z4	Посредством «Загрузчика» выгрузить модель
z5	Завершить работу «Загрузчика»
z6	Запомнить время вызова
z7	Установить смещение по времени «Частям»

Рис. 3. Граф переходов автомата «Модель» (A0)

3.3. Класс-автомат A1 («Загрузчик»)

3.3.1. Словесное описание

Реализует загрузку модели из двоичного файла, проверку корректности загрузки, высвобождение занятой памяти. Вызывается автоматом A0 «Модель» в выходных воздействиях z0, z1, z4, z5. Реализован, как потомок класса «LoggedAutomat».

3.3.2. Перечень состояний

0 – Не загружено (начальное состояние)
1 – Совершена попытка загрузки
2 – Загружено корректно
3 – Загружено с ошибкой

3.3.3. Перечень событий

4 – Требуется загрузить модель
5 – Требуется выгрузить модель
6 – Требуется проверить корректность загрузки

3.3.4. Перечень входных переменных

7 – Загружено без ошибок

3.3.5. Перечень выходных воздействий

8 – Произвести загрузку
9 – Произвести выгрузку

3.3.6. Схема связей

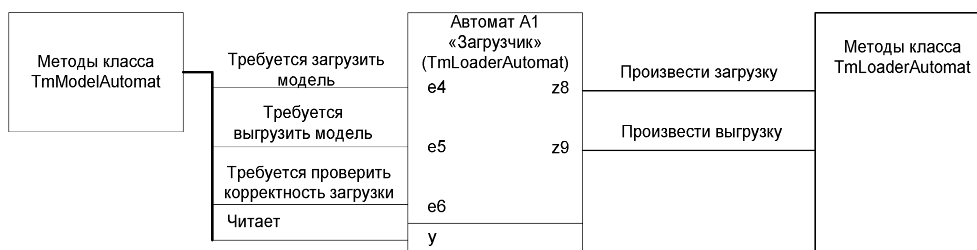
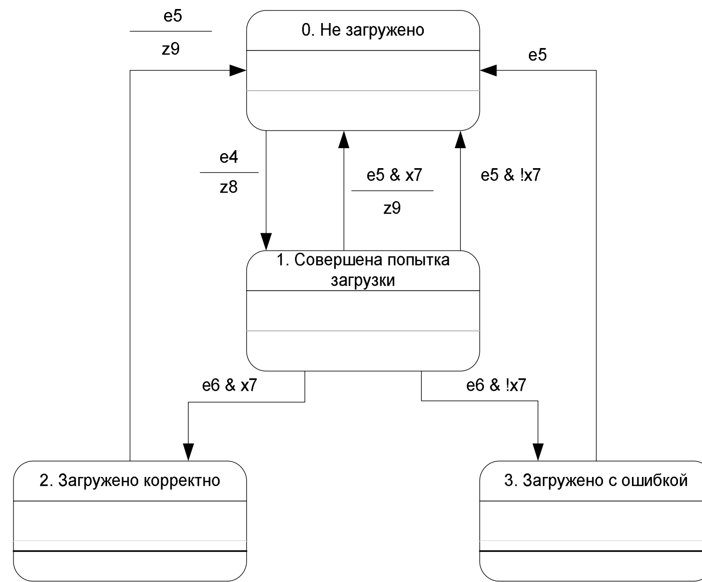


Рис. 4. Схема связей автомата «Загрузчик» (A1)

3.3.7. Граф переходов



e4	Требуется загрузить модель
e5	Требуется выгрузить модель
e6	Требуется проверить корректность загрузки

z8	Произвести загрузку
z9	Произвести выгрузку

x7	Загружено без ошибок

Рис. 5. Граф переходов автомата «Загрузчик» (A1)

3.4. Класс-автомат A2 («Часть»)

3.4.1. Словесное описание

Реализует поведение части модели. Содержит указатель на реализуемую часть, на всю модель и разницу по времени между последним вызовом и текущим. Реализован, как потомок класса «LoggedAutomat».

3.4.2. Перечень состояний

0 – Завершающий этап (начальное состояние)
1 – Собирается перейти в завершающий этап
2 – Обязательный этап

Обязательный этап – название для объединения предварительного и циклического этапов.

3.4.3. Перечень событий

0 – Прошло время

3.4.4. Перечень входных переменных

0 – После смещения без зацикливания часть останется в предварительном этапе
1 – Группу следует изменить
2 – Текущая группа приоритетнее следующей
3 – После перехода окажется в текущей группе
4 – Часть находится в обязательном этапе
11 – После перехода часть окажется вне обязательного этапа

3.4.5. Перечень выходных воздействий

5 – Рассчитать текущее положение кадра, находясь в обязательном этапе
6 – Запомнить текущий кадр и перейти в следующую группу
7 – Запомнить следующую группу
8 – Рассчитать текущее положение кадра, находясь в завершающем этапе
9 – Осуществить переход в новую группу по окончании текущей
10 – Осуществить переход в новую группу по окончании обязательного этапа текущей

3.4.6. Схема связей

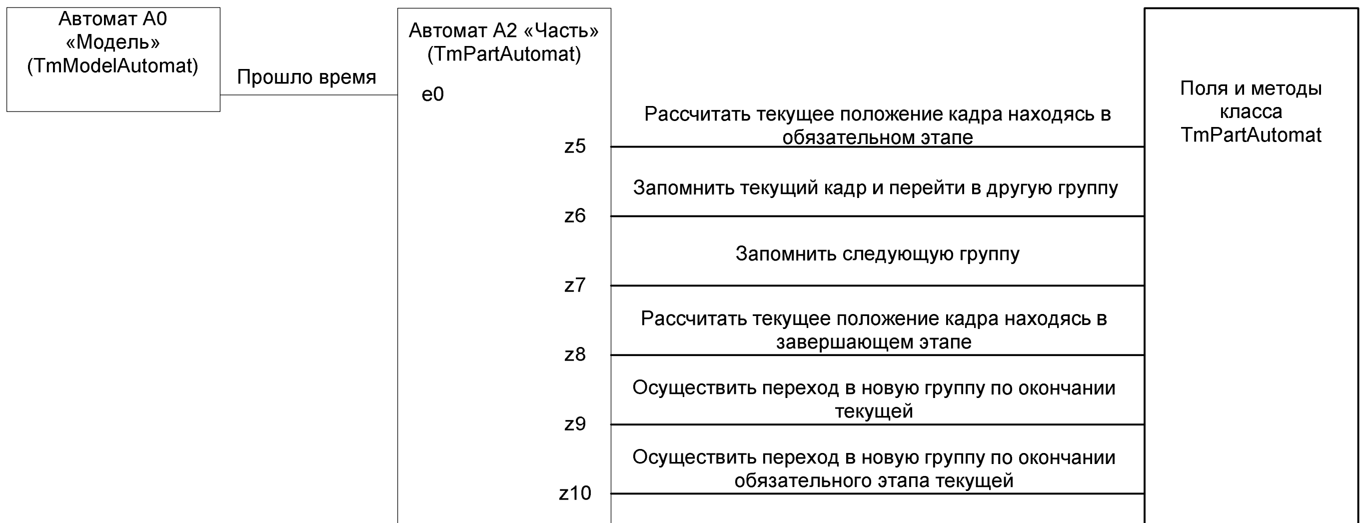
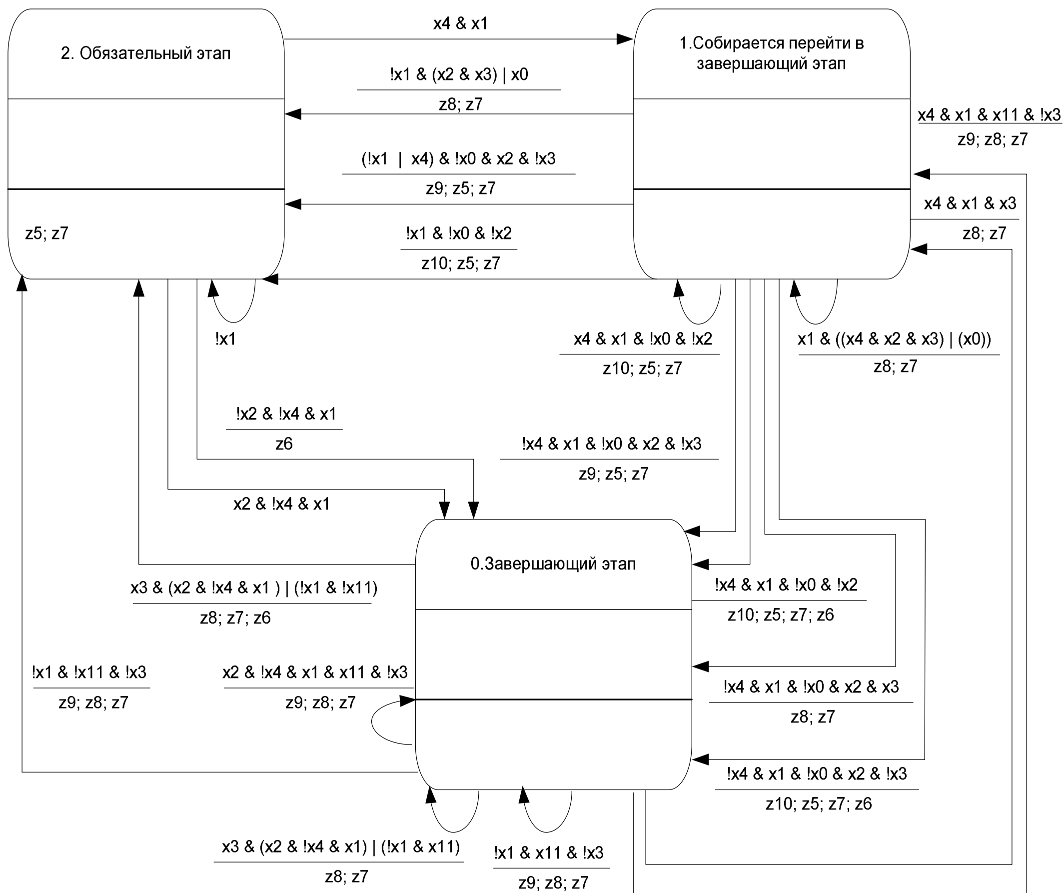


Рис. 6. Схема связей автомата «Часть» (A2)

3.4.7. Граф переходов



e0	Прошло время

z5	Рассчитать текущее положение кадра находясь в обязательном этапе
z6	Запомнить текущий кадр и перейти в другую группу
z7	Запомнить следующую группу
z8	Рассчитать текущее положение кадра находясь в завершающем этапе
z9	Осуществить переход в новую группу по окончании текущей
z10	Осуществить переход в новую группу по окончании обязательного этапа текущей

x0	После смещения без зацикливания останется в предварительной стадии или в цикле
x1	Группу следует изменить
x2	Текущая группа приоритетнее новой
x3	После перехода окажется в текущей группе
x4	Часть находится в обязательном этапе
x11	После перехода часть окажется вне обязательной группы

Рис. 7. Граф переходов автомата «Часть» (А2)

3.5. Класс «FileLogger»

3.5.1. Словесное описание

Реализует «составителя» протокола, позволяющего вести подробный, структурированный файл протокола. Реализован, как потомок класса «BaseLogger». Он переопределяет соответствующие методы.

3.6. Диаграмма классов

Классы, выделенные на диаграмме жирными линиями, содержатся в библиотеке «AutoLib»

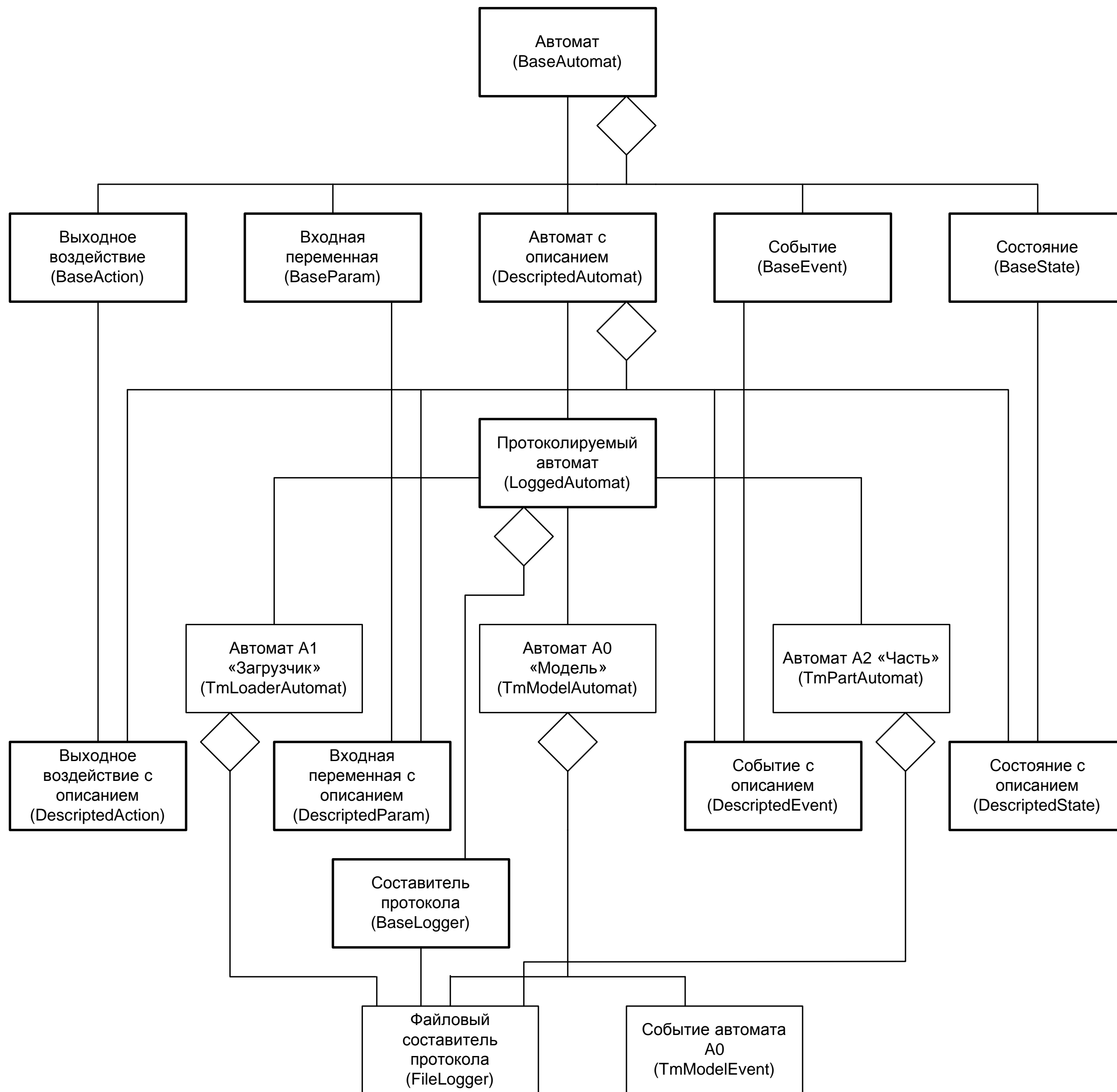


Рис. 8. Диаграмма классов

3.7. Схема взаимодействия автоматов

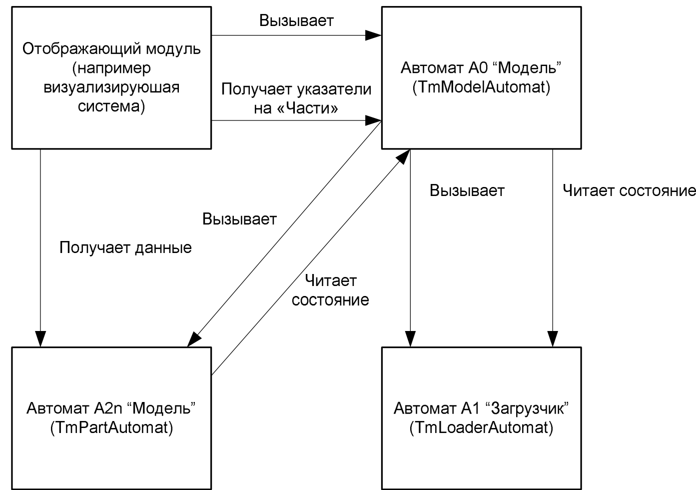


Рис. 9. Схема взаимодействия автоматов

Приложение содержит исходные тексты библиотеки «Auto-Lib» и всех автоматных классов. К сути работы мало относятся такие части исходного кода, как блок запуска программы и функции отображения, которые по этой причине в Приложении не приводятся.

Заключение

В настоящее время широко распространено объектно-ориентированное программирование. Сложно представить себе современного программиста, незнакомого с такими распространенными объектными библиотеками, как *VCL* и *MFC*. Библиотеки, написанные для таких языков, как *Java* и *C#*, в силу специфики языков, являются полностью объектными. Разумеется, у программиста при знакомстве с идеей, достойной быть примененной многократно (например, с автоматным подходом) в разных проектах или даже разными людьми, возникает желание представить автомат в виде объекта, отвечающего всем стандартам программистской «моды». Действительно, как излагается в книгах по объектно-ориентированному программированию, этот подход позволяет избежать многих проблем, таких как дублирование кода и сложность расширения, а также упрощает повторное использование кода. Совершенно естественным образом возникает идея создать объект «автомат».

В данном проекте продемонстрирован один из вариантов слияния автомата с объектом, что привело к появлению библиотеки «Auto-Lib». Приведен пример использования библиотеки.

В дальнейшем предполагается совершенствование библиотеки в части:

- повышения быстродействия;
- улучшения интерфейса;
- возможности множественного наследования автоматов;
- обработки исключений, например, перехватывание исключения и передача автомату специального события.

Источники

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. *Туккель Н.И., Шалыто А.А.* Танки и автоматы //BYTE/Россия. 2003. № 2. http://is.ifmo.ru, раздел «Статьи».
3. *Любченко В.С.* О билльярде с Microsoft Visual C++ 5.0 //Мир ПК. 1998. № 1.
4. *Шопырин Д.Г., Шалыто А.А.* Объектно-ориентированный подход к автоматному программированию. http://is.ifmo.ru, раздел «Проекты».
5. *Schoenblum Daniel E.* Спецификация формата *MD2*. http://linux.ucla.edu/~phaethon/q3/formats/md2-schoenblum.html
6. *Шалыто А.А.* Технологии автоматного программирования //Мир ПК. 2003. № 10. http://is.ifmo.ru, раздел «Статьи».

Приложение. Часть исходного кода

Библиотека «Auto-Lib»

```
// Auto-Lib.h
// Библиотека, предоставляющая набор классов, позволяющий
// легко создавать конечные автоматы и вести протокол событий,
// смен состояний, вычислений входных переменных и вызовов
// выходных воздействий

#ifndef __Auto_Lib_h
#define __Auto_Lib_h

#include <stdio.h>
#include "xString.h"
#include "xTypes.h"

//Объявление класса "Автомат"
class BaseAutomat;

//Объявление и реализация класс "Состояние"
class BaseState
{

public:

    uint32 y;      //Идентификатор состояния

    //Конструктор
    BaseState(uint32 y)
    {
           this->y=y;
    }

    //Конструктор копирования
    BaseState(BaseState &Y)
    {
           this->y=Y.y;
    }

    //Деструктор
    virtual ~BaseState()
    {

    }
};

//Объявление и реализация класса "Событие"
class BaseEvent
{

public:

  uint32 e;//Идентификатор события

    //Конструктор
    BaseEvent(uint32 e)
    {
           this->e=e;
    }

    //Деструктор
    virtual ~BaseEvent()
    {

    }
};

//Объявление и реализация класса "Входная переменная"
class BaseParam
{

public:

  uint32 x;//Идентификатор входной переменной
```

```
    //Конструктор
    BaseParam(uint32 x)
    {
           this->x=x;
    }

    //Деструктор
    virtual ~BaseParam()
    {

    }
};

//Объявление и реализация класса "Выходное воздействие"
class BaseAction
{

public:

  uint32 z;//Идентификатор выходного воздействия

    //Конструктор
    BaseAction(uint32 z)
    {
           this->z=z;
    }

    //Деструктор
    virtual ~BaseAction()
    {

    }
};

//Реализация класса "Автомат"
class BaseAutomat
{

public:

    BaseState *Y;  //Текущее состояние автомата(объект)
    uint32 y;             //Текущее состояние автомата(число)

    BaseEvent *E;  //Событие с которым вызывается автомат(объект)
    uint32 e;             //Событие с которым вызывается автомат(число)

    uint32 a;             //Идентификатор автомата

    BaseState **RegisteredStates; //Зарегистрированные события
    uint32 iRegisteredStatesNum;  //Количество зарегистрированных событий

    BaseEvent **RegisteredEvents; //Зарегистрированные события
    uint32 iRegisteredEventsNum;  //Количество зарегистрированных событий

    BaseParam **RegisteredParams; //Зарегистрированные входные переменные
    uint32 iRegisteredParamsNum;  //Количество зарегистрированных входных переменных

    BaseAction **RegisteredActions;      //Зарегистрированные выходные воздействия
    uint32 iRegisteredActionsNum; //Количество зарегистрированных выходных воздействий

    //Конструктор
    BaseAutomat(uint32 a)
    {
           this->Y=NULL;
           this->y=-1;
           this->a=a;
           iRegisteredStatesNum=0;
           iRegisteredEventsNum=0;
           iRegisteredParamsNum=0;
           iRegisteredActionsNum=0;
           RegisteredStates=new BaseState*[iRegisteredStatesNum];
           RegisteredEvents=new BaseEvent*[iRegisteredEventsNum];
           RegisteredParams=new BaseParam*[iRegisteredParamsNum];
           RegisteredActions=new BaseAction*[iRegisteredActionsNum];
    }

    //Деструктор
    virtual ~BaseAutomat()
    {
           uint32 i;
```

```
	for(i=0;i<iRegisteredStatesNum;i++)
	{
		delete RegisteredStates[i];
	}
	delete[] RegisteredStates;

	for(i=0;i<iRegisteredEventsNum;i++)
	{
		delete RegisteredEvents[i];
	}
	delete[] RegisteredEvents;

	for(i=0;i<iRegisteredParamsNum;i++)
	{
		delete RegisteredParams[i];
	}
	delete[] RegisteredParams;

	for(i=0;i<iRegisteredActionsNum;i++)
	{
		delete RegisteredActions[i];
	}
	delete[] RegisteredActions;
}

//Возвращает идентификатор текущего состояния
uint32 Get_y()
{
	return Y->y;
}

//Зарегистрировать состояние
virtual BaseState *RegisterState(uint32 y)
{
	BaseState *Y=new BaseState(y);
	AddState(Y);
	return Y;
}

//Добавить состояние в список состояний
void AddState(BaseState *Y)
{
	BaseState **NewRegisteredStates=new BaseState*[iRegisteredStatesNum+1];
	for(uint32 i=0;i<iRegisteredStatesNum;i++)
	{
		NewRegisteredStates[i]=RegisteredStates[i];
	}
	delete[] RegisteredStates;
	NewRegisteredStates[iRegisteredStatesNum]=Y;
	RegisteredStates=NewRegisteredStates;
	iRegisteredStatesNum++;
}

//Зарегистрировать событие
virtual BaseEvent *RegisterEvent(uint32 e)
{
	BaseEvent *E=new BaseEvent(e);
	AddEvent(E);
	return E;
}

//Добавить событие в список событий
void AddEvent(BaseEvent *E)
{
	BaseEvent **NewRegisteredEvents=new BaseEvent*[iRegisteredEventsNum+1];
	for(uint32 i=0;i<iRegisteredEventsNum;i++)
	{
		NewRegisteredEvents[i]=RegisteredEvents[i];
	}
	delete[] RegisteredEvents;
	NewRegisteredEvents[iRegisteredEventsNum]=E;
	RegisteredEvents=NewRegisteredEvents;
	iRegisteredEventsNum++;
}

//Зарегистрировать входную переменную
virtual BaseParam *RegisterParam(uint32 x)
{
	BaseParam *X=new BaseParam(x);
```

```
        AddParam(X);
        return X;
}

//Добавить входную переменную в список входных переменных
void AddParam(BaseParam *X)
{
        BaseParam **NewRegisteredParams=new BaseParam*[iRegisteredParamsNum+1];
        for(uint32 i=0;i<iRegisteredParamsNum;i++)
        {
                NewRegisteredParams[i]=RegisteredParams[i];
        }
        delete[] RegisteredParams;
        NewRegisteredParams[iRegisteredParamsNum]=X;
        RegisteredParams=NewRegisteredParams;
        iRegisteredParamsNum++;
}

//Зарегистрировать выходное воздействие
virtual BaseAction *RegisterAction(uint32 z)
{
        BaseAction *Z=new BaseAction(z);
        AddAction(Z);
        return Z;
}

//Добавить воздействие в список выходных воздействий
void AddAction(BaseAction *Z)
{
        BaseAction **NewRegisteredActions=new BaseAction*[iRegisteredActionsNum+1];
        for(uint32 i=0;i<iRegisteredActionsNum;i++)
        {
                NewRegisteredActions[i]=RegisteredActions[i];
        }
        delete[] RegisteredActions;
        NewRegisteredActions[iRegisteredActionsNum]=Z;
        RegisteredActions=NewRegisteredActions;
        iRegisteredActionsNum++;
}

//Реализация автомата (переопределяется в потомках)
void virtual Main()
{

}

//Вызов автомата с событием-объектом (реализован как не перегружаемая функция)
void BaseRun(BaseEvent *E)
{
        this->E=E;
    this->e=E->e;
        this->y=Y->y;
        uint32 old_y=y;
        BaseState *Old_Y=Y;
        Main();
        if (y!=old_y)
        {
                for(uint32 i=0;i<iRegisteredStatesNum;i++)
                {
                        if (RegisteredStates[i]->y==y)
                        {
                                Y=RegisteredStates[i];
                                return;
                        }
                }
                throw "Автомат перешел в некорректное состояние-число";
        } else
        {
                y=Y->y;
        }

}

//Вызов автомата с событием-объектом (реализован как перегружаемая функция)
void virtual Run(BaseEvent *E)
{
        BaseRun(E);
}

//Вызов автомата с событием-числом
```

```
    void virtual run(uint32 e)
    {
        for(uint32 i=0;i<iRegisteredEventsNum;i++)
        {
            if (RegisteredEvents[i]->e==e)
            {
                Run(RegisteredEvents[i]);
                return;
            }
        }
        throw "Вызов автомата с некорректным числовым значением события";
    }

    //Запуск вычисления n-ой входной переменной
    virtual bool x(uint32 n)
    {
        return Param(n);
    }

    //Реализация вычисление n-ой входной переменной
    virtual bool Param(uint32 n)
    {
        return false;
    }

    //Запуск n-го выходного воздействия
    virtual void z(uint32 n)
    {
        Action(n);
    }

    //Реализация n-го выходного воздействия
    virtual void Action(uint32 n)
    {

    }
};

//Объявление и реализация класса "Состояние с описанием"
class DescriptedState : public BaseState
{

public:

  xstring szStateDescription;    //Описание состояния

    //Конструктор
    DescriptedState(uint32 y,xstring szStateDescription) : BaseState(y)
    {
        this->szStateDescription=szStateDescription;
    }

    //Конструктор копирования
    DescriptedState(DescriptedState &Y) : BaseState(Y)
    {
        this->szStateDescription=Y.szStateDescription;
    }

    //Деструктор
    virtual ~DescriptedState()
    {

    }
};

//Класс "Событие с описанием"
class DescriptedEvent : public BaseEvent
{

public:

  xstring szEventDescription;    //Описание события

    //Конструктор
    DescriptedEvent(uint32 e,xstring szEventDescription) : BaseEvent(e)
    {
        this->szEventDescription=szEventDescription;
    }

    //Деструктор
```

```
    virtual ~DescriptedEvent()
    {

    }
};

//Класс "Входная переменная с описанием"
class DescriptedParam : public BaseParam
{

public:

  xstring szParamDescription;    //Описание входной переменной

    //Конструктор
    DescriptedParam(uint32 x,xstring szParamDescription) : BaseParam(x)
    {
           this->szParamDescription=szParamDescription;
    }

    //Деструктор
    virtual ~DescriptedParam()
    {

    }
};

//Класс "Выходное воздействие с описанием"
class DescriptedAction : public BaseAction
{

public:

  xstring szActionDescription;   //Описание выходного воздействия

    //Конструктор
    DescriptedAction(uint32 z,xstring szActionDescription) : BaseAction(z)
    {
           this->szActionDescription=szActionDescription;
    }

    //Деструктор
    virtual ~DescriptedAction()
    {

    }
};

//Объявление и реализация класса "Автомат с описанием"
class DescriptedAutomat : public BaseAutomat
{

public:

  xstring szAutomatDescription;  //Описание автомата

    //Конструктор
    DescriptedAutomat(uint32 a, xstring szAutomatDescription) : BaseAutomat(a)
    {
           this->szAutomatDescription=szAutomatDescription;
    }

    //Деструктор
    virtual ~DescriptedAutomat()
    {

    }

    //Зарегистрировать состояние с описанием
    virtual DescriptedState *RegisterState(uint32 y, xstring szStateDescription)
    {
           DescriptedState *Y=new DescriptedState(y,szStateDescription);
           AddState((BaseState*)Y);
           return Y;
    }

    //Зарегистрировать событие с описанием
    virtual DescriptedEvent *RegisterEvent(uint32 e, xstring szEventDescription)
    {
           DescriptedEvent *E=new DescriptedEvent(e,szEventDescription);
```

```
        AddEvent((BaseEvent*)E);
        return E;
    }

    //Зарегистрировать входную переменную с описанием
    virtual DescriptedParam *RegisterParam(uint32 x, xstring szParamDescription)
    {
        DescriptedParam *X=new DescriptedParam(x,szParamDescription);
        AddParam((BaseParam*)X);
        return X;
    }

    //Зарегистрировать выходное воздействие с описанием
    virtual DescriptedAction *RegisterAction(uint32 z, xstring szActionDescription)
    {
        DescriptedAction *Z=new DescriptedAction(z,szActionDescription);
        AddAction((BaseAction*)Z);
        return Z;
    }
};

//Объявление и реализация класса "Составитель протокола"
class BaseLogger
{

public:

    //Конструктор
    BaseLogger()
    {

    }

    //Деструктор
    virtual ~BaseLogger()
    {

    }

    //Запротоколировать создание автомата
    virtual LogAutomatCreated(DescriptedAutomat *A)
    {

    }

    //Запротоколировать завершение работы автомата
    virtual LogAutomatFinished(DescriptedAutomat *A)
    {

    }

    //Запротоколировать регистрацию состояния
    virtual LogStateRegistered(DescriptedAutomat *A,DescriptedState *Y)
    {

    }

    //Запротоколировать регистрацию события
    virtual LogEventRegistered(DescriptedAutomat *A,DescriptedEvent *Y)
    {

    }

    //Запротоколировать регистрацию входной переменной
    virtual LogParamRegistered(DescriptedAutomat *A,DescriptedParam *X)
    {

    }

    //Запротоколировать регистрацию выходного воздействия
    virtual LogActionRegistered(DescriptedAutomat *A,DescriptedAction *Z)
    {

    }

    //Запротоколировать изменение состояния
    virtual LogStateChanged(DescriptedAutomat *A, DescriptedState *Old_Y,DescriptedState *Y)
    {

    }
```

```
    //Запротоколировать начало обработки события
    virtual LogEventRaised(DescriptedAutomat *A,DescriptedEvent *E)
    {

    }

    //Запротоколировать окончание обработки события
    virtual LogEventFinished(DescriptedAutomat *A,DescriptedEvent *E)
    {

    }

    //Запротоколировать начало вычисление входной переменной
    virtual LogParamCalcStarted(DescriptedAutomat *A, DescriptedParam *X)
    {


    }

    //Запротоколировать окончание вычисление входной переменной
    virtual LogParamCalcFinished(DescriptedAutomat *A, DescriptedParam *X, bool x)
    {


    }

    //Запротоколировать начало вызова выходного воздействия
    virtual LogActionStarted(DescriptedAutomat *A, DescriptedAction *Z)
    {


    }

    //Запротоколировать окончание вызова выходного воздействия
    virtual LogActionFinished(DescriptedAutomat *A, DescriptedAction *Z)
    {


    }

    //Запротоколировать произвольную строку
    virtual LogString(xstring szComment)
    {

    }

};

//Объявление и реализация класса "Протоколируемый автомат"
class LoggedAutomat : public DescriptedAutomat
{

public:

  BaseLogger *Logger;     //Составитель протокола

    //Конструктор
    LoggedAutomat(uint32 a,xstring szAutomatDescription,BaseLogger *Logger) :
DescriptedAutomat(a,szAutomatDescription)
    {
            this->Logger=Logger;
            if (Logger!=NULL)Logger->LogAutomatCreated(this);
    }

    //Деструктор
    ~LoggedAutomat()
    {
            if (Logger!=NULL)Logger->LogAutomatFinished(this);
    }

    //Зарегистрировать состояние с описанием и занести в протокол
    virtual DescriptedState* RegisterState(uint32 y,xstring szStateDescription)
    {
            DescriptedState *Y=new DescriptedState(y,szStateDescription);
            AddState((BaseState*)Y);
            if (Logger!=NULL)Logger->LogStateRegistered((DescriptedAutomat*)this,Y);
            return Y;
    }
```

```
//Зарегистрировать событие с описанием и занести в протокол
virtual DescriptedEvent* RegisterEvent(uint32 e,xstring szEventDescription)
{
	DescriptedEvent *E=new DescriptedEvent(e,szEventDescription);
	AddEvent((BaseEvent*)E);
	if (Logger!=NULL)Logger->LogEventRegistered((DescriptedAutomat*)this,(DescriptedEvent*)E);
	return E;
}

//Зарегистрировать входную переменную с описанием и занести в протокол
virtual DescriptedParam* RegisterParam(uint32 x,xstring szParamDescription)
{
	DescriptedParam *X=new DescriptedParam(x,szParamDescription);
	AddParam((BaseParam*)X);
	if (Logger!=NULL)Logger->LogParamRegistered((DescriptedAutomat*)this,X);
	return X;
}

//Зарегистрировать выходное воздействие с описанием и занести в протокол
virtual DescriptedAction* RegisterAction(uint32 z,xstring szActionDescription)
{
	DescriptedAction *Z=new DescriptedAction(z,szActionDescription);
	AddAction((BaseAction*)Z);
	if (Logger!=NULL)Logger->LogActionRegistered((DescriptedAutomat*)this,(DescriptedAction*)Z);
	return Z;
}


//Вызвать автомат с событием-объектом
void virtual Run(BaseEvent *E)
{
	if (Y==NULL) Y=RegisteredStates[0];
	if (Logger!=NULL)Logger->LogEventRaised((DescriptedAutomat*)	this,(DescriptedEvent*)E);
	DescriptedState Old_Y=*(DescriptedState*)Y;
	BaseRun(E);
	if (Logger!=NULL)Logger->LogStateChanged((DescriptedAutomat*)this,&Old_Y,(DescriptedState*)Y);
	if (Logger!=NULL)Logger->LogEventFinished((DescriptedAutomat*)this,(DescriptedEvent*)E);
}

//Запуск вычисления n-ой входной переменной и занесение в протокол
virtual bool x(uint32 n)
{
	BaseParam *X=NULL;
	for(uint32 i=0;i<iRegisteredParamsNum;i++)
	{
		if (RegisteredParams[i]->x==n)
		{
			X=RegisteredParams[i];
			break;
		}
	}
	if (X==NULL)
	{
		throw "Вычисление входной переменной с некорректным номером";
	}
	if (Logger!=NULL)Logger->LogParamCalcStarted((DescriptedAutomat*)this,(DescriptedParam*)X);
	bool b=Param(n);
	if (Logger!=NULL)Logger->LogParamCalcFinished((DescriptedAutomat*)this,(DescriptedParam*)X,b);
	return b;
}

//Запуск n-го выходного воздействия и занесение в протокол
virtual void z(uint32 n)
{
	BaseAction *Z=NULL;
	for(uint32 i=0;i<iRegisteredActionsNum;i++)
	{
		if (RegisteredActions[i]->z==n)
		{
			Z=RegisteredActions[i];
			break;
		}
	}
	if (Z==NULL)
	{
		throw "Вычисление входной переменной с некорректным номером";
	}
	if (Logger!=NULL) Logger->LogActionStarted((DescriptedAutomat*)this,(DescriptedAction*)Z);
	Action(n);
	if (Logger!=NULL) Logger->LogActionFinished((DescriptedAutomat*)this,(DescriptedAction*)Z);
```

```
    }
};
#endif
```

Реализация автомата «Модель» (A0)

```
void TmModelAutomat::Main()
{
    if (y==300000)
    {
            if(e==6)
            {
                    z(0);
                    z(1);
                    z(2);
                    y=0;
                    return;
            }
    }

    //Любое состояние готовности
    if (y!=300000)
    {
            if (e==7)
            {
                    z(3);
                    z(4);
                    z(5);
                    y=300000;
                    return;
            }

            //Следующий блок реализует переход в одно из 32768 состояний
            if (e==8)
            {
                    //Переход в другое состояние
                    for(uint32 i=1;i<16;i++)
                    {
                            if (x(i) && x(0)) y|=(1<<(i-1));
                            if (x(i) && !x(0)) y&=~(1<<(i-1));
                    }

                    //Протоколирование
                    Y->y=y;
                    DescriptedState *DS=(DescriptedState*)Y;
                    DS->szStateDescription="Готов[";
                    for(uint8 j=0;j<15;j++)
                    {
                            if (y&(1<<j)) DS->szStateDescription+=Caption[j]+" ";
                    }
                    DS->szStateDescription+="]";

                    z(7);
                    //Делегирование события всем частям
                    for(int i=0;i<PartCount;i++)
                    {
                            A2[i]->run(0);
                    }
                    z(6);
            }
    }
}
```

Реализация автомата «Загрузчик» (A1)

```
void TmLoaderAutomat::Main()
{
    switch(y)
    {
    case 0:
            if (e==4)
            {
                    z(8);
                    y=1;
            }
            break;
    case 1:
            if (e==6 && x(7))

            {
                    y=2;
            }
            if (e==6 && !x(7))

            {
                    y=3;
            }
            if (e==5 && x(7))

            {
                    z(9);
                    y=0;
            }
            if (e==5 && !x(7))

            {
                    y=0;
            }
            break;
    case 2:
            if (e==5)
            {
                    z(9);
                    y=0;
            }
            break;
    case 3:
            if (e==5)
            {
                    y=0;
            }
                    break;
    }
}
```

Реализация автомата «Часть» (A2)

```
void TmPartAutomat::Main()
{
    switch(y)
    {
    case 0:
            if (x(4) && x(1) && x(3))
            {
                    z(8);
                    z(7);
                    y=1;
            } else

            if (x(3) && ((x(2) && !x(4) && x(1)) || (!x(1) && x(11))))
            {
                    z(8);
                    z(7);
                    y=0;
            } else

            if (x(3) && ((!x(1) && !x(11)) || (!x(2) && !x(4) && x(1))))
            {
```

```
                z(8);
                z(7);
                z(6);
                y=2;
        } else

        if (x(4) && x(1) && x(11) && !x(3))
        {
                z(9);
                z(8);
                z(7);
                y=1;
        } else

        if(x(2) && !x(4) && x(1) && x(11) && !x(3))
        {
                z(9);
                z(8);
                z(7);
                y=0;
        } else

        if (!x(1) && !x(11) && !x(3))
        {
                z(9);
                z(8);
                z(7);
                y=2;
        } else

        if (!x(1) && x(11) && !x(3))
        {
                z(9);
                z(8);
                z(7);
                y=0;
        } else

case 1:
        if (x(1) &&((x(4) && x(2) && x(3))||x(0)))
        {
                z(8);
                z(7);
                y=1;
        } else

        if (!x(1) && ((x(2) && x(3))||x(0)))
        {
                z(8);
                z(7);
                y=2;
        } else


        if(!x(4) && x(1) && !x(0) && x(2) && x(3))
        {
                z(8);
                z(7);
                y=0;
        } else

        if ((!x(1) || x(4)) && !x(0) && x(2) && !x(3))
        {
                z(9);
                z(5);
                z(7);
                y=2;
        } else

        if(!x(4) && x(1) && !x(0) && x(2) && !x(3))
        {
                z(9);
                z(5);
                z(7);
                y=0;

        } else

        if (x(4) && x(1) && !x(0) && !x(2))
        {
```

```
                z(10);
                z(5);
                z(7);
                y=1;
        } else

        if (!x(4) && x(1) && !x(0) && !x(2))
        {
                z(10);
                z(5);
                z(7);
                z(6);
                y=0;
        } else

        if (!x(1) && !x(0) && !x(2))
        {
                z(10);
                z(5);
                z(7);
                y=2;
        }
        break;

    case 2:
        z(5);
        z(7);

        if (x(4) && x(1))
        {
                y=1;
        } else

        if(x(2) && !x(4) && x(1))
        {
                y=0;
        } else

        if (!x(2) && !x(4) && x(1))
        {
                z(6);
                y=0;
        } else

        if (!x(1))
        {
                y=2;
        }
        break;
    }
}
```

Пример фрагмента протокола

```
Протоколирование начато
{
   A0(Модель) создан
   A0(Модель) вызвано e6(Подготовиться к работе)
      A0(Модель) начал выполнять z0(Создать 'Загрузчик')
         A1(Загрузчик) создан
      A0(Модель) выполнил z0(Создать 'Загрузчик')
      A0(Модель) начал выполнять z1(Посредством 'Загрузчика' загрузить модель, проверить корректность)
         A1(Загрузчик) вызвано e4(Требуется загрузить модель)
            A1(Загрузчик) начал выполнять z8(Произвести загрузку)
            A1(Загрузчик) выполнил z8(Произвести загрузку)
            A1(Загрузчик) y0(Не загружено)-->y1(Совершена попытка загрузки)
         A1(Загрузчик) обработал e4(Требуется загрузить модель)
         A1(Загрузчик) вызвано e6(Требуется проверить корректность загрузки)
            A1(Загрузчик) начал вычислять x7(Загружено без ошибок)
            A1(Загрузчик) вычислил x7(Загружено без ошибок): [True]
            A1(Загрузчик) начал вычислять x7(Загружено без ошибок)
            A1(Загрузчик) вычислил x7(Загружено без ошибок): [True]
            A1(Загрузчик) y1(Совершена попытка загрузки)-->y2(Загружено корректно)
         A1(Загрузчик) обработал e6(Требуется проверить корректность загрузки)
      A0(Модель) выполнил z1(Посредством 'Загрузчика' загрузить модель, проверить корректность)
      A0(Модель) начал выполнять z2(Создать 'Части')
         A20(Часть-0) создан
```

```
      A21(Часть-1) создан
   A0(Модель) выполнил z2(Создать 'Части')
   A0(Модель) y300000(Не готов к работе)-->y0(Готов - начальное)
A0(Модель) обработал e6(Подготовиться к работе)
A0(Модель) вызвано e8(Активировано новое действие)
   A0(Модель) начал вычислять x1(Действие - атака-1)
   A0(Модель) вычислил x1(Действие - атака-1): [False]
   A0(Модель) начал вычислять x1(Действие - атака-1)
   A0(Модель) вычислил x1(Действие - атака-1): [False]
```