САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ МЕХАНИКИ И ОПТИКИ

Кафедра «Компьютерных Технологий»

А. В. Ларионов

РАЗРАБОТКА ВИЗУАЛЬНОГО ЯЗЫКА АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

на основе инструментов для создания специализированных языков предметной области среды разработки *Microsoft Visual Studio 2005*

Санкт-Петербург
2005

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время в среде разработчиков программного обеспечения прослеживается тенденция к сближению спецификации, модели и исходного кода программного продукта. Данная тенденция получила свое выражение в подходе названном *Model Driven Architecture* (*MDA*) [1]. Все чаще *UML*-диаграммы [2] используются в качестве спецификаций, одновременно описывая и модель поведения. По *UML*-моделям генерируется исходный код. Такое сближение позволяет минимизировать риски возникновения неверной интерпретации спецификаций и вследствие этого, написания реализации неверной функциональности, а также минимизируются риски возникновения ошибок и дефектов.

Среда разработки *Microsoft Visual Studio* [3] является основной средой разработки программных продуктов для семейств операционных систем *Microsoft Windows*. В версии 2005 года этой среды разработки наблюдается стремление к визуализации процесса проектирования и разработки программных продуктов. В ней появилась возможность проектирования распределенных систем с помощью *Дизайнера Распределенных Систем*, возможность создания классов с помощью *Дизайнера Классов*, а также возможность создания собственных визуальных редакторов для описания различных моделей и генерации кода по моделям.

Одной из широко используемых моделей для представления динамики исполнения «реактивных» систем является модель конечного автомата. В данной работе описывается пример добавления инструментального средства проектирования автоматов в существующую среду разработки *Microsoft Visual Studio 2005* таким образом, чтобы упростить процесс разработки «реактивных» систем и в то же время позволить добавлять подобные системы в существующие проекты.

1. ПОСТАНОВКА ЗАДАЧИ

Разработать и реализовать инструментальное средство автоматного программирования, интегрированного в среду разработки *Microsoft Visual Studio 2005*.

Инструментальное средство должно позволять разрабатывать автоматные системы визуальным способом и полноценно расширять возможности *Microsoft Visual Studio 2005* по проектированию и реализации программных продуктов.

Инструментальное средство должно преобразовывать визуальное представление конечного автомата в исходные коды на языке программирования *C#* [4, 5]. В качестве событий, по которым следует производить переходы из состояния в состояние, следует использовать события языка *C#.* В качестве защитных условий на переходах следует использовать условия языка *C#.* В качестве действий, производимых по переходам, на входе в состояния и выходе из состояний, следует использовать операторы языка *C#.*

В инструментальное средство следует включить возможность преобразование модели конечного автомата в исходные коды на других языках программирования, совместимых с *CLR (Common Language Runtime)*.

Инструментальное средство должно предусматривать возможность не только создания новых программных продуктов содержащих автоматные системы, но и позволять добавлять автоматное поведение в существующие исходные коды приложений.

2. ОБЗОР ПРОГРАММНЫХ ПРОДУКТОВ – АНАЛОГОВ

В данной главе рассмотрены некоторые программные продукты позволяющие визуально проектировать автоматные модели. К сожалению, почти все они не позволяют генерировать код по модели, и модель используется лишь как часть спецификации разрабатываемой программы.

Более-менее законченным средством для разработки автоматных моделей и генерации исполняемого кода по ним является программный продукт *SmartState* от *ApeSoft* (http://www.smartstatestudio.com/). На рис. 1 представлена визуальная модель автомата в *SmartState*.

Рис. 1. Диаграмма автомата в *SmartState*

Но и *SmartState* не лишен многих недостатков:

- отсутствие какой-либо связи со статической моделью программы;
- отсутствие валидации модели;
- код, генерируемый по модели, не соответствует стилю программирования на языке C#;
- ограниченность в передаче параметров при обработке событий;
- необходимость использования внешней библиотеки для обработки автоматов в разрабатываемой программе.

Далее приведен список программных продуктов, которые хоть и позволяют разрабатывать автоматные модели поведение отдельных объектов программы, но не поддерживают генерацию кода для автоматных моделей:

- *Metamill* от *Metamill Software* (http://www.metamill.com/);
- *Enterprise Architect Professional* от *Sparx Systems* (http://www.sparxsystems.com/);
- *Rose XDE Developer for Visual Studio* от *IBM-Rational* (http://www.ibm.com/software/rational/);
- *Visual Paradigm Suite* от *Visual Paradigm* (http://www.visual-paradigm.com/product/vpsuite/).

Существует также бесплатный преобразователь языка автоматов в исходные коды различных языков программирования *The State Machine Compiler*. Однако, само представление и редактирование автоматной модели в виде текстового языка не удобно.

Бесплатный *.NET State Machine Toolkit*, который можно найти на сайте *CodeProject*, позволяет создавать автоматные системы с помощью таблиц состояний и переходов, что тоже не особо удобно.

Также существует программный продукт под названием *Windows Workflow Foundation*, входящий в состав *WinFX* от *Microsoft*. Сейчас он находится на стадии разработки, но уже можно узнать некоторые возможности, которые будут поддерживаться. *Windows Workflow Foundation* позволит создавать не только

Диаграммы Состояний, но и *Диаграммы Активностей* и *Диаграммы Последовательностей* и другие диаграммы. Также позволит валидировать диаграмму и генерировать по ней соответствующий исходный код, и будет возможен процесс визуальной отладки диаграмм.

Существует также ряд программных продуктов для разработки автоматных систем и последующего их преобразования в исходный код на языке *Java*, такие как бесплатный *Unimod* (http://unimod.sourceforge.net/) и платный *AnyLogic* от *XJ Technologies* (http://www.xjtek.com/). Эти продукты включают в себя множество возможностей для упрощения процесса разработки программного обеспечения.

3. РЕАЛИЗАЦИЯ

При выполнении данной работы были использованы *Инструменты Специализированных Языков Предметной Области* (*Domain-Specific Language Tools*, *DSL Tools*) [6] входящие в состав *Microsoft Visual Studio 2005 SDK*. Ниже приведены общее описания понятия специализированного языка предметной области и *DSL Tools*.

Специализированный язык предметной области (*Domain-Specific Language*, DSL) – это язык, разработанный для того, чтобы быть полезным для решения узкого специфического круга задач, в отличие от языков общего применения. Используя инструменты *DSL Tools,* могут быть созданы специализированные инструменты моделирования путем определения нового языка моделирования и его реализации. Например, возможно создание специализированного языка для описания интерфейса пользователя, бизнес процесса, базы данных или потоков информации и последующая генерации кода из этого описания.

Инструменты Специализированных Языков Предметной Области могут быть использованы для построения индивидуальных визуальных редакторов приспособленных к любой предметной области. Например, используя *Инструменты Специализированных Языков Предметной Области*, возможно создание инструмента для описания концепций моделирования бизнес процессов характерных для определенной организации. При создании инструмента диаграмм состояний, описывается, что такое состояние, свойства состояния, типы состояний, определяются переходы между состояниями и т.д. Диаграмма состояний, описывающая статус контрактов в страховой компании, и диаграмма состояний, описывающая взаимодействие пользователя со страницами Web-сайта, внешне (поверхностно) похожи, но лежащие в их основе концепции значительно различаются. Создавая собственный специализированный язык предметной области и индивидуальный редактор модели, появляется возможность точно

определить, какие концепции диаграммы состояний нужно использовать в данном инструменте.

В работе *Инструменты Специализированных Языков Предметной Области* были использованы для создания визуального языка автоматного программирования.

3.1. Создание DSL проекта

Процесс разработки нового специализированного языка предметной области начинается с использования мастера “Создания Специализированного Языка Предметной Области”. Мастер предусматривает выбор шаблона проекта, расширения файлов языка и других свойств проекта.

В данной работе был выбран шаблон *Минимального Языка* и расширение файлов языка `.statemachine`.

В результате работы мастера получается проект *Visual Studio* содержащий следующие модули:

- Метамодель Предметной Области (`Domain Model`) – содержит описание классов и взаимоотношений языка, и код, генерируемый из описания. Файл `DomainModel.dsldm` используется для визуального изменения метамодели предметной области. Кроме того, этот модуль используется для определения валидации моделей специализированного языка предметной области путем написания кода.
- Редактор (`Designer`) – содержит в файле `Designer.dsldd` определения графических представлений, используемых редактором, их отображение в концепции метамодели предметной области, и код, генерируемый из этих определений. Модуль редактора также содержит файлы ресурсов для обеспечения картинок и текстовых строк. Кроме того, модуль содержит параметры использования

системы валидации и, при необходимости, код для изменения поведения редактора модели.

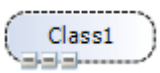
Основной код *DSL* проекта, кроме пользовательских изменений, генерируется по файлам, определяющим язык – `DomainModel.dsldm` и `Designer.dsldd`. Код генерируется из текстовых шаблонов, читающих определения языка и выводящих соответствующий код. Поэтому после модификации файлов определяющих язык, следует запустить команду *Трансформации Всех Шаблонов (Transform All Templates)* нажатием кнопки на панели *Проводника Проекта*.

Тестирование специализированного языка предметной области проводится путем запуска построенного проекта. При запуске проекта происходит загрузка экспериментального экземпляра *Microsoft Visual Studio 2005* с проектом отладки. Проект отладки позволяет тестировать специализированный язык предметной области. Для того чтобы использовать специализированный язык предметной области для создания кода или других артефактов, предусмотрено написание текстовых шаблонов, читающих модель и генерирующих файлы из нее. После проверки корректности работы специализированного языка предметной области, возможно создание пакета развертывания (файл `.msi`) для распространения языка.

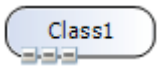
3.2. Разработка метамодели предметной области

Метамодель предметной области разрабатывается путем визуальной модификации файла `DomainModel.dsldm`. В графическим редакторе метамодели приняты следующие графические обозначения:

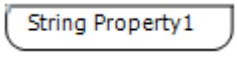

– корневой элемент метамодели;



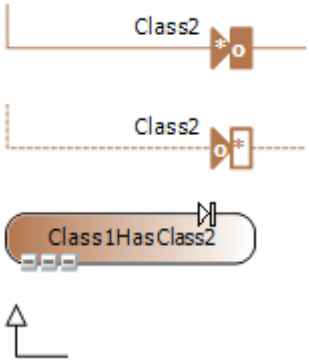
– абстрактный класс;



– класс;



– свойство класса;



– отношение встраивания;

– отношение ссылки;

– класс отношения;

– наследование.

Отношения также включают в себя две роли: роль исходного и целевого классов. Роли определяют множественность элементов в отношении и имена свойств добавляемых в классы для обеспечения доступа к объектам классов участников отношения.

3.2.1. Начальная метамодель

Сначала была разработана метамодель предметной области, приведенная на рис. 2.

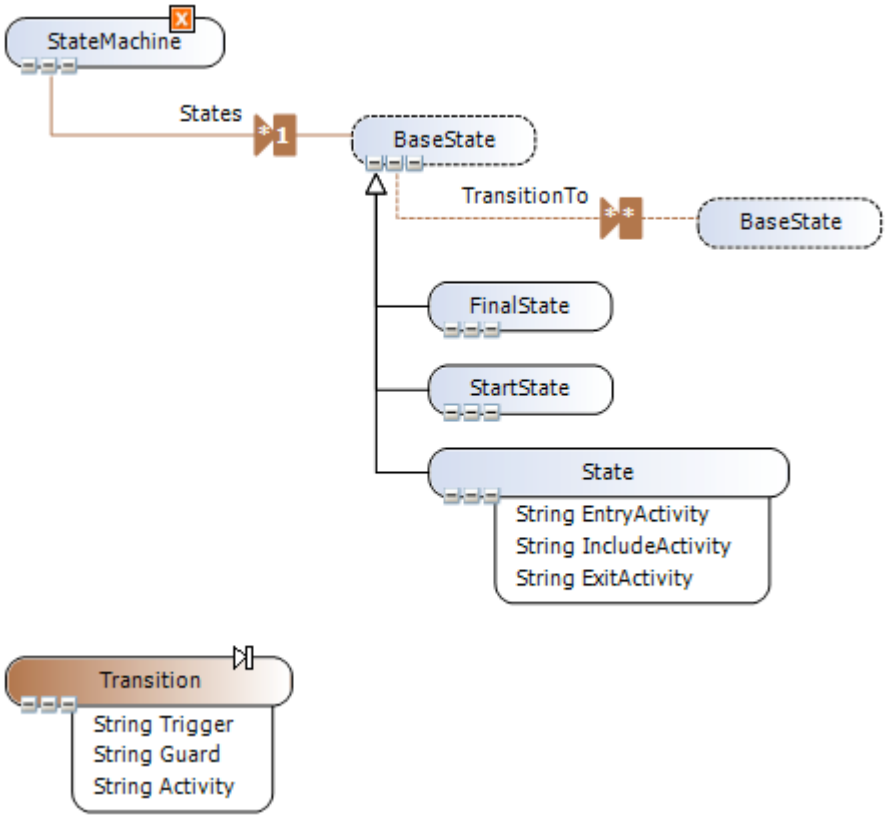


Рис. 2. Первично разработанная метамодель

Словесно метамодель можно описать следующим образом. Корневым элементом метамодели является класс `StateMachine` (конечный автомат). Класс `StateMachine` состоит (отношение встраивания) из `BaseState` (базовых состояний). Абстрактный класс `BaseState` является базовым для классов `FinalState` (конечное состояние), `StartState` (начальное состояние), `State` (состояние). Класс `State` имеет три текстовых свойства: `EntryActivity` (действие на входе в состояние), `IncludeActivity` (действие в состоянии), `ExitActivity` (действие на выходе из состояния). Переходы между состояниями отношение ссылки `Transition` (переход) между объектами класса `BaseState`. Класс отношения `Transition` имеет три текстовых свойства: `Trigger` – событие, по которому следует производить переход, `Guard` – условие которое должно быть выполненным для перехода, `Activity` – действие на переходе.

Для описания событий, условий и действий были выбраны текстовые свойства для прямого соответствия артефактам (событиям, условиям, операторам) целевого языка.

3.2.2. Изменения метамодели для редактора

В связи с ограничениями редактора модели, существующими в используемой версии *DSL Tools*, указанными в разд. 3.4.1, в метамодель предметной области были внесены необходимые изменения. Полученная метамодель приведена на рис. 3, изменения выделены желтым цветом.

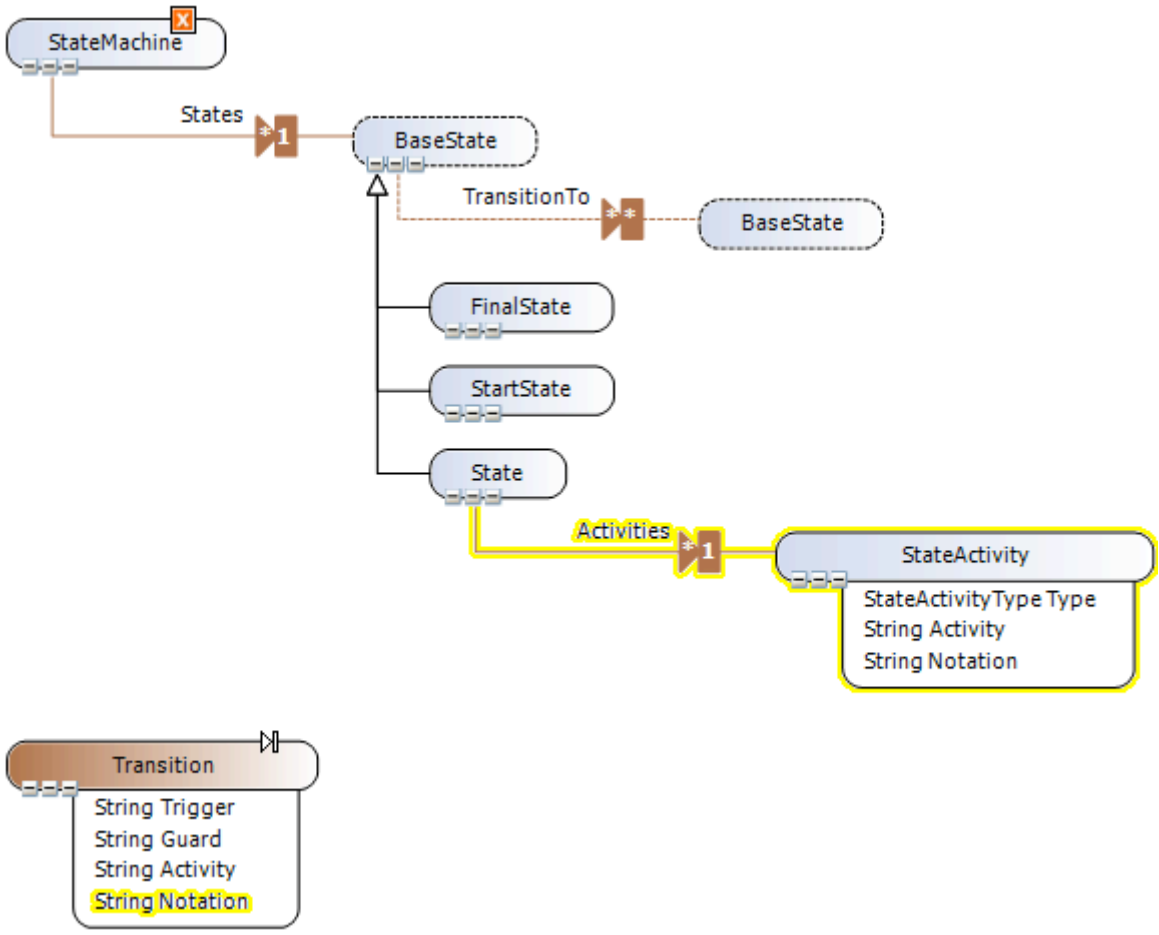


Рис. 3. Метамодель, модифицированная для редактора

В метамодель добавился класс `StateActivity` (действие в состоянии). Класс `StateActivity` имеет следующие свойства: `Type` (тип) перечислимого типа, определяющий тип активности (на входе в состояние, внутри состояния, на выходе из состояния), `Activity` (действие), `Notation` (запись). Класс `State` потерял все свои текстовые свойства, но стал связан с классом `StateActivity` отношением встраивания. В класс `Transition` добавилось текстовое свойство `Notation` (запись).

Текстовые свойства `Notation` классов `Transition` и `StateActivity` используются для визуального представления свойств соответствующих классов и не могут быть изменены из редактора.

3.2.3. Изменения метамодели для механизма валидации

На рис. 4 изображена метамодель, измененная для поддержки валидации модели. Изменения выделены желтым цветом.

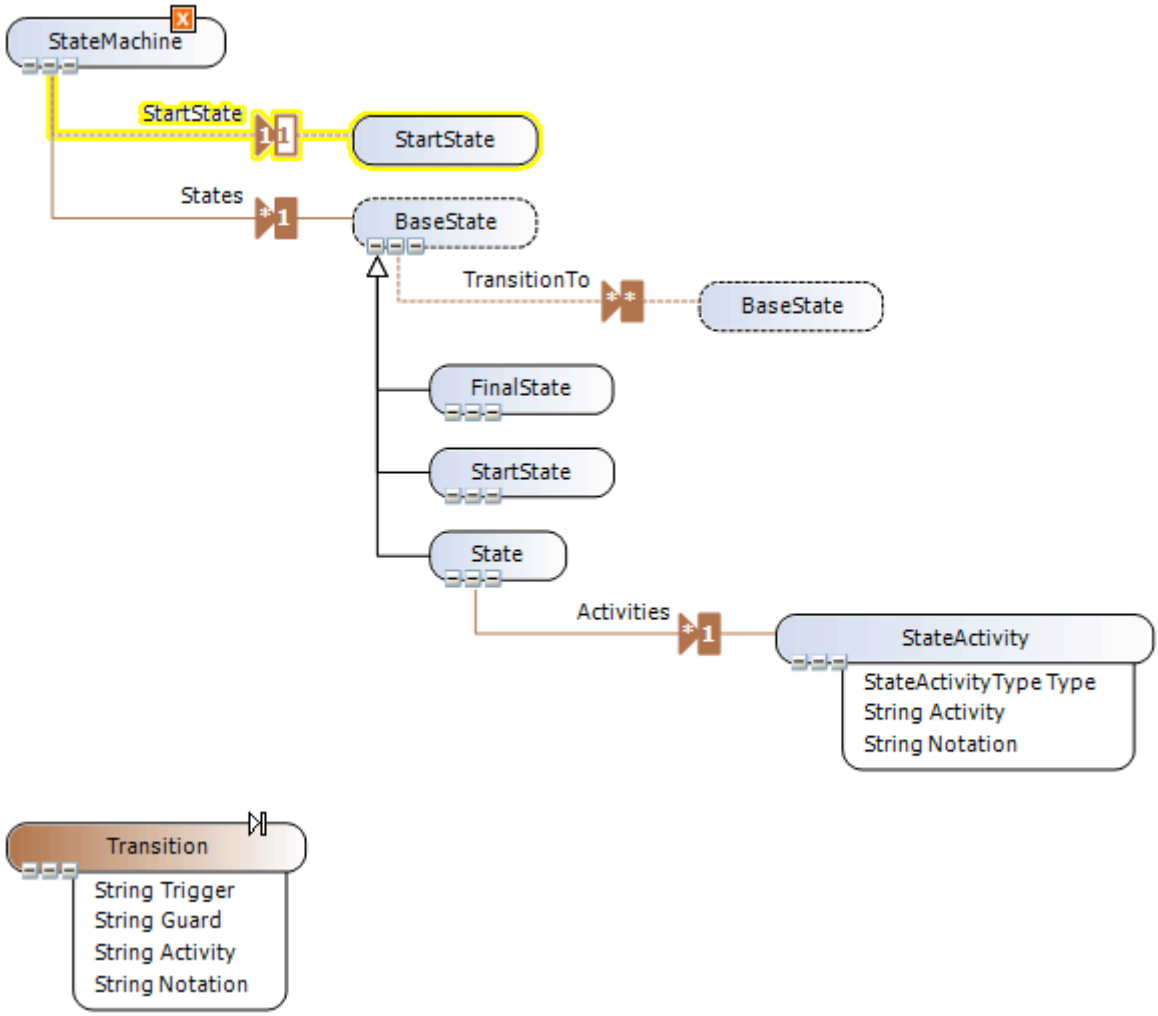


Рис. 4. Метамодель, модифицированная для механизма валидации

В метамодель было добавлено отношение ссылки между классами `StateMachine` и `StartState`.

3.2.4. Изменения метамодели для генерации кода

На рис. 5 изображена метамодель, измененная для поддержки генерации кода по модели. Изменения выделены желтым цветом.

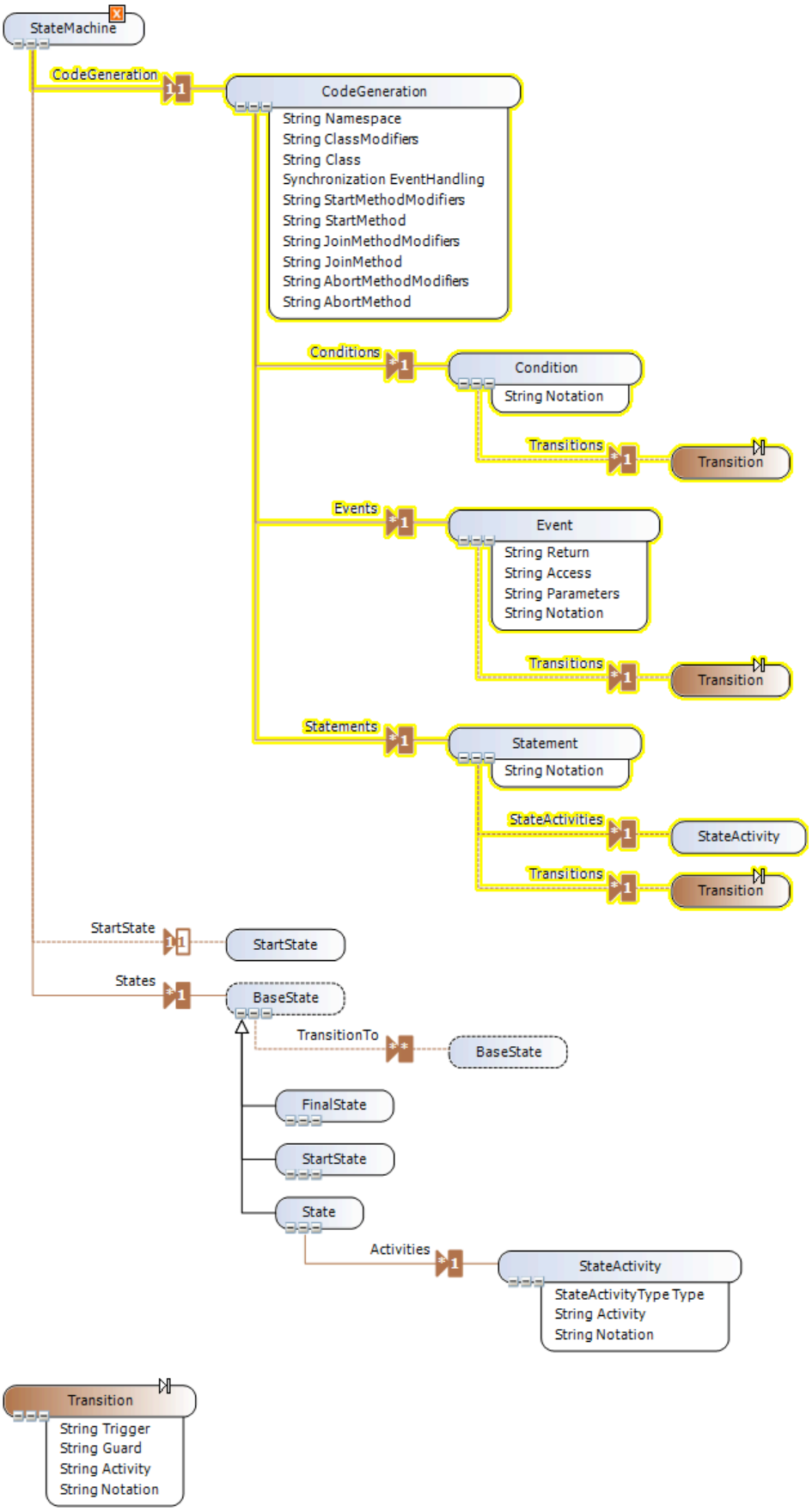


Рис. 5. Метамодель, модифицированная для генерации кода

В метамодель были добавлены следующие классы:

- `CodeGeneration` (генерация кода) – класс определяющий свойства генерируемого кода такие как namespace (пространство имен), имя класса автомата, модификаторы класса автомата, выбор синхронной или асинхронной обработки событий, имя методов запуска автомата, прерывания работы автомата, ожидания окончания работы автомата и их модификаторы;
- `Event` (событие) – класс инкапсулирующий событие в терминах целевого языка (доступ к событию, возвращаемое значение, параметры);
- `Condition` (условие) – класс инкапсулирующий условие в терминах целевого языка;
- `Statement` (оператор) – класс инкапсулирующий оператор в терминах целевого языка.

В метамодель были добавлены отношения встраивания между классами `StateMachine` и `CodeGeneration`, `CodeGeneration` и `Event`, `Condition` и `Statement`. Также были добавлены отношение ссылки между классами `Event` и `Transition`, `Condition` и `Transition`, `Statement` и `Transition`, `Statement` и `StateActivity`.

3.3. Разработка правил изменения модели

Для поддержки валидности модели и представления ее в редакторе *DSL Tools* позволяет добавлять правила изменения модели. Правила могут быть выполнены при добавлении нового объекта класса метамодели в модель, при удалении объекта, при изменении свойств объекта. Эти правила реализуются путем написания классов наследуемых от абстрактных классов правил и перегрузкой методов, вызываемых при соответствующем действии. Атрибутом *RuleOn* определяется, на каком классе метамодели будет выполняться правило. Базовым классом определяется, на каком действии будет вызвано правило. Также

правила должны быть зарегистрированы в частичном (*partial*) классе *GeneratedMetaModelTypes* путем создания в нем статического поля инициализированном типом класса правила.

3.3.1. Правила для редактора

В данном разделе описаны правила изменения модели, которые были добавлены в связи с ограничениями редактора и поддержания валидности представления модели в редакторе языка:

- при добавлении объекта класса `State` в модель добавляются три объекта класса `StateActivity` и связываются с ним. Свойства `Type` объектов класса `StateActivity` устанавливаются в значения `Entry`, `Include`, `Exit`;
- при попытке добавления объекта класса `StateActivity` в объект класса `State` проверяется, что объект класса `State` содержит менее трех объектов класса `StateActivity`, иначе в добавлении отказывается;
- при попытке удаления объекта класса `StateActivity` из объекта класса `State` проверяется, что включающий его объект класса `State` удаляется, иначе в удалении отказывается;
- при изменении свойства `Activity` объекта класса `StateActivity` свойство `Notation` обновляется к виду "**`type/activity`**", где **`type`** – текстовое значение свойства `Type`, **`activity`** – значение свойства `Activity`;
- при изменении свойств `Trigger`, `Guard` или `Activity` объекта класса `Transition` свойство `Notation` обновляется к виду "**`trigger[guard]/activity`**", где **`trigger`** – значение свойства `Trigger`, **`guard`** – значение свойства `Guard`, **`activity`** – значение свойства `Activity`.

3.3.2. Правила для механизма валидации

В данном разделе описаны правила изменения модели, которые были добавлены для упрощения механизма валидации модели:

- при попытке добавления объекта класса `StartState` в объект класса `StateMachine` проверяется, что объект класса `StateMachine` не содержит уже объект класса `StartState` путем проверки свойства `StartState` объекта класса `StateMachine` на равенство нулевой ссылке, иначе в добавлении отказывается. При добавлении свойству `StartState` объекта класса `StateMachine` присваивается значение ссылки на объект класса `StartState`;
- при удалении объекта класса `StartState` из объекта класса `StateMachine`, свойству `StartState` объекта класса `StateMachine` присваивается значение нулевой ссылки.

3.3.3. Правила для генерации кода

В данном разделе описаны правила изменения модели, которые были добавлены для поддержания валидности представления свойств генерации кода в редакторе языка:

- при создании объекта класса `StateMachine` – при первом открытии файла с расширением `.statemachine` созданного по шаблону, объект класса `CodeGeneration` создается и добавляется в объекта класса `StateMachine`;
- при попытке удаления объекта класса `CodeGeneration` из объекта класса `StateMachine` в удалении отказывается;
- при изменении свойства `Activity` объекта класса `StateActivity` происходит связывание объекта `StateActivity` с объектом класса `Statement`, свойство `Notation` которого совпадает со свойством `Activity` объекта класса `StateActivity`. Если свойство

Activity объекта класса StateActivity становится пустым, то связь разрывается. Если соответствующий объект класса Statement не найден, то он создается;

- при изменении свойства Trigger объекта класса Transition происходит связывание объекта Transition с объектом класса Event, свойство Access которого совпадает со свойством Trigger объекта класса Transition. Если свойство Trigger объекта класса Transition становится пустым, то связь разрывается. Если соответствующий объект класса Event не найден, то он создается;
- при изменении свойства Guard объекта класса Transition происходит связывание объекта Transition с объектом класса Condition, свойство Notation которого совпадает со свойством Guard объекта класса Transition. Если свойство Guard объекта класса Transition становится пустым, то связь разрывается. Если соответствующий объект класса Condition не найден, то он создается;
- при изменении свойства Activity объекта класса Transition происходит связывание объекта Transition с объектом класса Statement, свойство Notation которого совпадает со свойством Activity объекта класса Transition. Если свойство Activity объекта класса Transition становится пустым, то связь разрывается. Если соответствующий объект класса Statement не найден, то он создается;
- при изменении свойств Return, Access или Parameters объекта класса Event свойство Notation обновляется к виду "**return access(parameters)**", где **return** – значение свойства Return, **access** – значение свойства Access, **parameters** – значение свойства Parameters;

- при изменении свойства `Access` объекта класса `Event` свойству `Trigger` всех объектов класса `Transition`, на которые ссылается объекта класса `Event`, присваивается значение свойства `Access`;
- при попытке удаления объекта класса `Event` из объекта класса `CodeGeneration` проверяется, что объект класса `Event` не содержит ссылок ни на один объект класса `Transition`, иначе в удалении отказывается;
- при изменении свойства `Notation` объекта класса `Condition` свойству `Guard` всех объектов класса `Transition`, на которые ссылается объекта класса `Condition`, присваивается значение свойства `Notation`;
- при попытке удаления объекта класса `Condition` из объекта класса `CodeGeneration` проверяется, что объект класса `Condition` не содержит ссылок ни на один объект класса `Transition`, иначе в удалении отказывается;
- при изменении свойства `Notation` объекта класса `Statement` свойству `Activity` всех объектов класса `Transition` и всех объектов класса `StateActivity`, на которые ссылается объекта класса `Statement`, присваивается значение свойства `Notation`;
- при попытке удаления объекта класса `Statement` из объекта класса `CodeGeneration` проверяется, что объект класса `Statement` не содержит ссылок ни на один объект класса `Transition` и ни на один объект класса `StateActivity`, иначе в удалении отказывается.

3.4. Разработка визуального редактора языка

В используемой версии *DSL Tools* разработка визуального редактора производится путем внесения изменений в *XML* файл `Designer.dsldd`. Визуальный редактор этого файла отсутствует. В последующих версиях *DSL*

Tools такой редактор появится. Полная версия этого файла находится в приложении, а в данной главе словесно описаны основные значимые элементы.

Визуальный редактор путем изменения графической диаграммы позволяет модифицировать модель, которой соответствует объект корневого класса метамодели. Для того чтобы представить объекты классов метамодели и классов отношений метамодели, в визуальном редакторе используются графические представления, которые можно добавлять на диаграмму с помощью инструментария.

Существуют несколько нотаций для графического описания конечных автоматов, такие как приняты в *UML 2.0* [2, 7], SWITCH-технологии [8] и у Харела [9]. В данной работе было использована нотация близкая к *UML 2.0*, с некоторыми незначительными изменениями.

3.4.1. Графические представления

В редакторе языка следующие графические представления изображаются на диаграмме с помощью пиктограмм:

● – “Начальное Состояние”;

◉ – “Конечное Состояние”.

Изначально предполагалось реализовать графическое представление, отражающее “Состояние” как прямоугольник со скругленными углами. Внутри сверху добавить имя, а внутри в середине три строковых поля отражающих действия в состоянии. К сожалению, в используемой версии *DSL Tools* невозможно добавить несколько текстовых полей одно под другим, а лишь возможно указать одно из девяти внутренних позиций или восьми внешних. Поэтому графическое представление “Состояния” было реализовано, как секционное (рис. 6) и были внесены соответствующие изменения в метамодель предметной области.

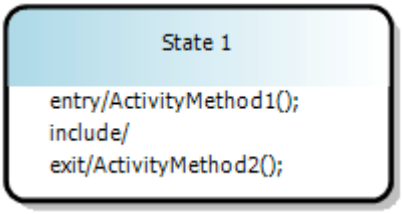


Рис. 6. Графическое представление “Состояния”

В единственной секции графического представления “Состояния” выводятся свойство `Notation` объектов класса `StateActivity`, вложенных в объект класса `State`.

На рис. 7 показано графическое представление “Перехода”, которое может исходить только из графических представлений “Начальное Состояние” и “Состояние”, а входить только в графические представления “Состояние” и “Конечное Состояние”.

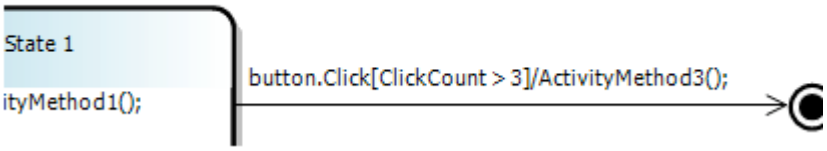


Рис. 7. Графическое представление “Перехода”

Над началом графического представления “Перехода” выводится свойство `Notation` объекта класса `Transition`.

3.4.2. Связи с классами предметной области

Визуальный редактор позволяет модифицировать модель, которой соответствует объект корневого класса метамодели. Таким образом, при первом открытии файла с расширением `.statemachine`, полученного по шаблону, создается объект класса `StateMachine`.

При добавлении графических представлений на диаграмму происходит создание объектов соответствующих классов метамодели и добавление этих объектов в объект класса `StateMachine`. В и классов метамодели.

таблица 1 приведено соответствие графических представлений и классов метамодели.

Таблица 1

“Начальное Состояние”	Класс `StartState` метамодели
“Состояние”	Класс `State` метамодели
“Конечное Состояние”	Класс `FinalState` метамодели
“Переход”	Класс `Transition` метамодели

При добавлении графического представления “Перехода” на диаграмму происходит добавление объекта класса отношения `Transition` метамодели и связывание двух объектов класса `BaseState` по средствам объекта класса отношения `Transition`.

3.4.3. Инструментарий визуального редактора языка

В инструментарий входят четыре инструмента:

- *Start State* – инструмент добавления графического представления “Начального Состояния” на диаграмму, создание объекта класса `StartState` и встраивание его в объект класса `StateMachine`;
- *State* – инструмент добавления графического представления “Состояния” на диаграмму, создание объекта класса `State` и встраивание его в объект класса `StateMachine`;
- *Final State* – инструмент добавления графического представления “Конечного Состояния” на диаграмму, создание объекта класса `FinalState` и встраивание его в объект класса `StateMachine`;
- *Transition* – инструмент добавление графического представления “Перехода” на диаграмму и добавления отношения ссылки между двумя объектами класса `BaseState`.

3.4.4. Графического представление “Генерации Кода”

Описание графического представления “Генерации Кода” вынесено в отдельную главу, так как соответствующий ей класс `CodeGeneration` метамодели не имеет прямого отношения к самой модели автомата, а определяет генерацию кода автомата. Более того, для графического представления “Генерации Кода” нет инструмента добавления, она автоматически добавляется при выполнении правила создания объекта класса `StateMachine` и её нельзя удалить, так как это запрещено правилом удаления объекта класса `CodeGeneration`.

Графическое представление “Генерация Кода” реализована, как секционное. В трех секциях выводятся свойства `Notation` вложенных в объект класса `CodeGeneration` объектов классов `Event`, `Condition` и `Statement` (рис. 8).

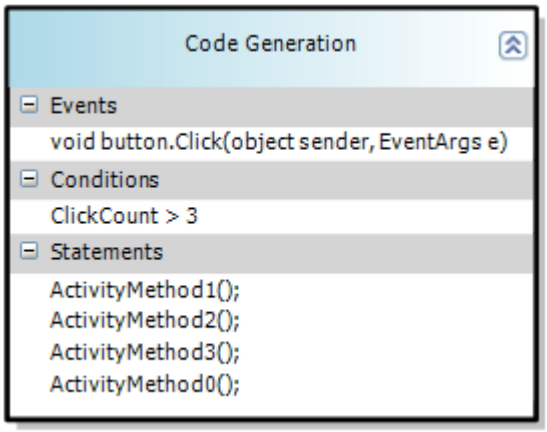


Рис. 8. Графическое представление “Генерации Кода”

3.5. Разработка механизма валидации модели

Поддержка механизма валидации модели в *DSL Tools* реализована следующим образом. Для каждого класса метамодели пишутся валидирующие методы. Обработчик *DSL Tools* находит методы валидации с помощью проверки наличия атрибута `ValidationMethod` у метода. В параметрах атрибута следует указать, когда данный метод валидации следует вызывать: на открытии файла, на сохранении, по вызову команды валидации из контекстного меню.

Ниже словесно описаны методы валидации для классов метамодели языка.

3.5.1. Валидация объекта класса `StateMachine`

Метод валидации класса `StateMachine` проверяет уникальность имен всех состояний, проверяет наличие и единственность начального состояния и проверяет достижимость состояний из начального и достижимость конечных состояний из всех состояний, если конечные состояния есть в модели.

3.5.2. Валидация объекта класса `BaseState`

Метод валидации класса `BaseState` проверяет, что при существовании более одного исходящего перехода по одному и тому же событию, все переходы имеют защитные условия и для всех переходов производимых по одному и тому же событию защитные условия не могут быть выполнены вместе.

Так как сама подобная проверка не являлась основной для данной работы, а также сложна с теоретический точки зрения и для практической реализации, в данном методе валидации реализован лишь сбор строковых защитных условий для переходов по одному событию и вызов заглушки метода проверки на ортогональность условий.

3.5.3. Валидация объекта класса `StartState`

Метод валидации класса `StartState` проверяет отсутствие входящих переходов.

3.5.4. Валидация объекта класса `State`

Метод валидации класса `State` проверяет присутствие трех объектов класса `StateActivity` со свойством `Type`, равным значениям соответственно: `Entry`, `Include`, `Exit`.

3.5.5. Валидация объекта класса `FinalState`

Метод валидации класса `FinalState` проверяет отсутствие исходящих переходов.

3.5.6. Валидация объекта класса `Transition`

Метод валидации класса `Transition` проверяет, что если переход исходит из начального состояния, то его событие не определено, в противном случае событие должно быть определено.

3.5.7. Валидация объекта класса `CodeGeneration`

Метод валидации класса `CodeGeneration` проверяет валидность указанных свойств пространства имен, имени класса автомата, модификаторов и имен методов автомата. Также проверяется валидность используемых событий, условий и операторов из свойств вложенных объектов классов `Event`, `Condition` и `Statement`.

Проверка валидности строковых артефактов целевого языка и проверка ортогональность условий производится с помощью статических методов класса `Artefacts`. Класс `Artefacts` был специально добавлен в проект языка для этих целей. Однако, так как проверка валидности строковых артефактов целевого языка не является основной для данной работы, методы класса `Artefacts` оставлены пустыми и всегда возвращают результат успешной проверки.

3.6. Разработка шаблона для генерации кода

В используемой версии *DSL Tools* генерация кода по модели производится путем добавления в проект содержащий файл модели языка файлов текстовых шаблонов, при обработке которых происходит генерация кода по модели. Так как данный вариант генерации кода не удобен для разработки (при изменении файла модели нужно выполнять команду *Трансформации Всех Шаблонов*), был разработан собственный генератор кода наследованный от генератора кода по текстовому шаблону. Разработанный генератор кода загружает текстовый шаблон из ресурсов модуля и подставляет в текст шаблона нужные имена файлов для корректной генерации кода по модели. Текст шаблона генерации кода приведен в **Error! Reference source not found.**Рис. 8.

При выполнении модели автомата происходит логгирование всех изменений связанных с данной моделью, используя вызов стандартного метода логгирования `System.Diagnostics.Trace.WriteLine`. Протоколируются следующие события:

- запуск выполнения модели автомата;
- входы в состояния;
- действия на входе в состояния;
- выходы из состояний;
- действия на выходе из состояний;
- остановка выполнения модели автомата;
- обработка событий воздействующих на модель автомата;
- вычисление защитных условий на переходах;
- действия на переходах.

Как можно было заметить из описания свойств класса `CodeGeneration` метамодели, генерируемый код автомата для обработки событий бывает двух различных типов: синхронная обработка и асинхронная. При синхронной обработке событий выполнение проверки защитного условия, выполнение действия на выходе из состояния, выполнение действия на переходе и выполнение действия на входе в состояние происходят прямо в обработчике события. При асинхронной обработке событий идентификатор события и параметры события сохраняются в очередь, а извлекаются они из очереди в отдельном потоке. Однако при обоих вариантах генерации кода гарантируется безопасность вызовов методов класса из нескольких потоков, обработка только одного события в один момент времени и минимизация риска обработки событий пришедших из разных потоков вне очередности.

Также следует отметить, что все имена методов (кроме имен методов для управления автоматом, объявленных в объекте класса `CodeGeneration`) и полей генерируемого класса автомата являются уникальными за счет добавления к их именам буквенно-цифрового глобально-уникального идентификатора. Это

позволяет совмещать класс автомата с уже имеющимся классом в проекте с помощью модификатора `partial`.

3.6.1. Синхронная обработка событий

В связи с синхронной обработкой событий, на действия, выполняемые при обработке события (проверки защитного условия, действия на выходе из состояния, действия на переходе и действия на входе в состояние), налагается ограничение, что эти действия не порождают события обрабатываемые автоматом.

Основной задачей генерации кода синхронной обработки событий стала минимизация риска обработки событий вне очередности. Конструкция языка *C#* `lock ( … ) { … }` гарантирует, что весь код заключенный в фигурные скобки будет выполняться только одним потоком, остальные же потоки войдя в эту конструкцию будут дожидаться освобождения блокировки. Однако эта конструкция не гарантирует очередность разблокировки ожидающих потоков.

Эта задача была решена с помощью создания очереди потоков породивших события обрабатываемые автоматом. При обработке события поток сначала добавляет свой идентификатор в конец очереди и ждет, пока его идентификатор окажется первым в очереди. Далее поток обрабатывает событие, а на выходе из обработчика удаляет свой идентификатор из начала очереди и снимает блокировку со ждущих потоков для того, чтобы каждый поток проверил свою очередь на выполнение. Поток, идентификатор которого оказывается в начале очереди, начинает обрабатывать событие:

```
// Field contains queue of event threads for sync StateMachine
private Queue<Thread> eventQueue = new Queue<Thread>();

private void OnEvent( object sender, EventArgs e )
{
    lock ( eventQueue )
    {
        if ( ( 0 < eventQueue.Count ) &&
             ( Thread.CurrentThread == eventQueue.Peek() ) )
            throw new InvalidOperationException( "Processed event can't produce events!" );

        eventQueue.Enqueue( Thread.CurrentThread );
        while ( Thread.CurrentThread != eventQueue.Peek() )
            Monitor.Wait( eventQueue );
    }

    try
    {
```

```
                HandleEvent( sender, e );
            }
            finally
            {
                lock ( eventQueue )
                {
                    if ( Thread.CurrentThread != eventQueue.Dequeue() )
                        throw new InvalidOperationException( "Abnormal event processing! This exception
must never been thrown!" );
                    Monitor.PulseAll( eventQueue );
                }
            }
        }
```

3.6.2. Асинхронная обработка событий

При асинхронной обработке событий параметры события, передаваемые в его обработчик, сохраняются вместе с идентификатором события в очередь. После этого обработчик события возвращает управление потоку, породившему событие. Следовательно, параметры, переданные по ссылке, могут быть модифицированы. В связи со всем вышесказанным, на параметры, передаваемые по ссылке в обработчики событий, накладываются ограничение о неизменности используемых свойств при обработке события. Так как при выполнении автомата невозможно определить изменились ли свойства параметра переданного по ссылке или нет, то исполнение этого ограничения полностью ложится на разработчика программного продукта.

4. ОПИСАНИЕ РАЗРАБОТАННОГО ИСТРУМЕНТАЛЬНОГО СРЕДСТВА

Разработанное инструментальное средство интегрируется в среду разработки *Microsoft Visual Studio 2005*. На рис. 9 показана среда разработки во время редактирования автомата.

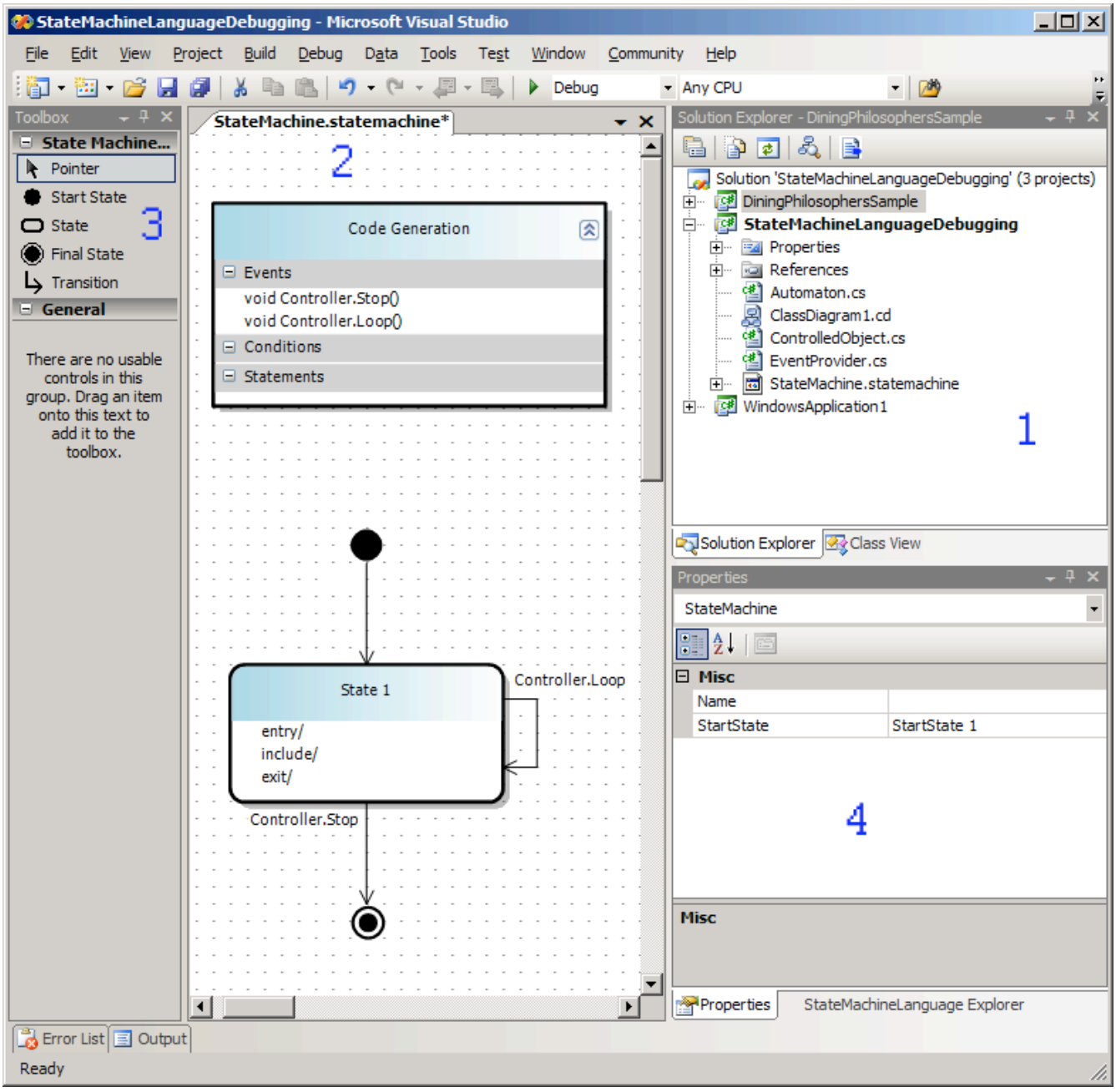


Рис. 9. *Microsoft Visual Studio 2005* – редактирование автомата

Цифрами синим цветом отмечены основные окна среды разработки.

1. *Проводник Проекта* – древовидный список модулей и вложенных в них файлов проекта. Используя команду “Add New Item…” *Проводника Проекта* в проект можно добавлять новые файлы языка автоматов `.statemachine`.
2. Редактируемый файл языка автомата `.statemachine`. В него добавляются графические представления “Состояний”, “Переходов”, а так же в нем существует единственное графическое представление “Генерации кода”.
3. Инструментарий редактора языка автоматов. Позволяет добавлять соответствующие графические представления в редактируемый файл.
4. Окно свойств активного объекта среды разработки.

5. ПРИМЕНЕНИЕ РАЗРАБОТАННОГО ИСТРУМЕНТАЛЬНОГО СРЕДСТВА

В результате разработанного проекта специализированного языка автоматного программирования было получено инструментальное средство автоматного программирования, интегрированное в среду разработки *Microsoft Visual Studio 2005*. Оно позволяет разрабатывать программные продукты, используя автоматный подход, а также к уже имеющимся исходным кодам приложений добавлять автоматное поведение для отдельных объектов.

Ниже приведены примеры использования.

5.1. Изменение текста на кнопке

Данный пример показывает общие черты использования инструментального средства. Здесь приводится решение простейшей задачи реакции на нажатие кнопки окна. При нажатии текст на кнопке должен меняться с “Opened” на “Closed” и обратно. При нажатии на кнопку увеличивается счетчик. Если на кнопку нажимали более пяти раз и текст на ней “Opened”, то при очередном нажатии на кнопку текст должен измениться на “Dead”. Далее нажатие на кнопку не должно приводить ни к каким действиям.

Создаем форму и кладем на кнопку с текстом “Dummy” (рис. 10).

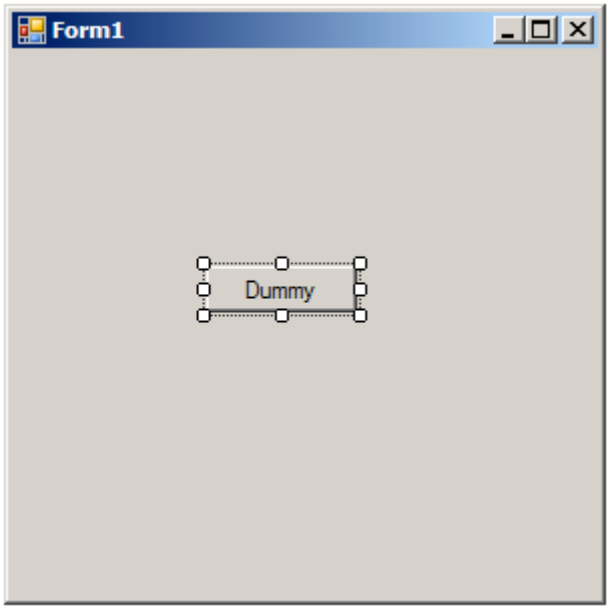


Рис. 10. Дизайн формы

Добавляем в проект новую диаграмму классов и в редакторе классов создаем новый класс `ButtonClicker`, добавляем в него также поле `button` типа `System.Windows.Forms.Button`, поле `count` целого типа, свойство целого типа только для чтения `ClickCount`, методы `Open`, `Close`, `IncrementCount`, `Die` и конструктор с параметром `button` типа `System.Windows.Forms.Button` (рис. 11). При визуальном добавлении методов при помощи диаграммы классов, в исходные коды соответствующих классов добавляются пустые методы-заглушки.

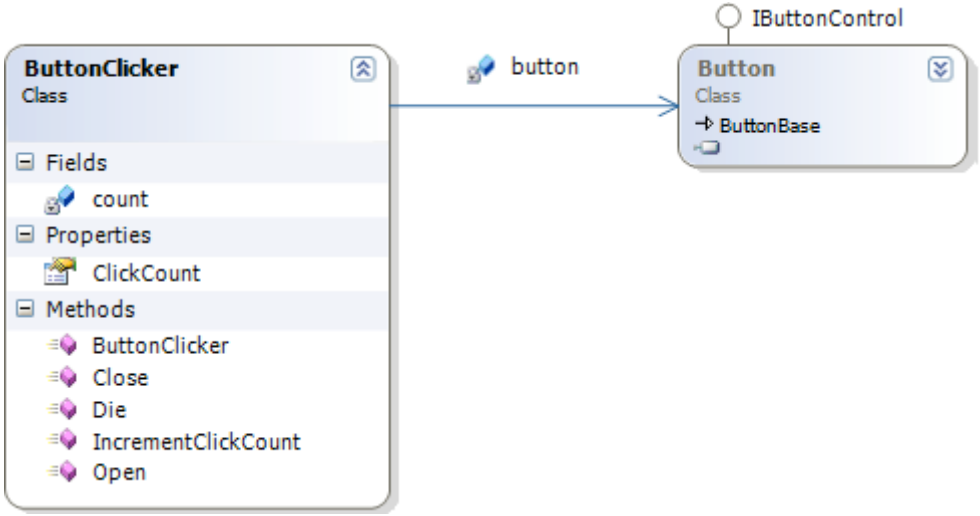


Рис. 11. Диаграмма классов

Добавляем в проект новый файл автоматного языка `ButtonClicker.statemachine`. Устанавливает свойство `Custom Tool` этого файла в значение “`StateMachineGenerator`”. Это требуется для генерации кода. Рисуем в нем следующую диаграмму: начальное состояние, состояние “`Opened`”, состояние “`Closed`”, конечное состояние и переходы. На входе в состояние “`Opened`” вызываем метод `Open`, на входе в состояние “`Closed`” вызываем метод `Close`. На переходах из состояния “Opened” в состояние “Closed” по событию `button.Click` вызываем метод `IncrementClickCount`. На переходе из состояния “Open” в конечное состояние вызываем метод `Die`. Устанавливаем защитные условия на переходах из состояния “Opened” в состояние “Closed” и конечное состояние в значения `ClickCount <= 5` и `ClickCount >= 5` соответственно.

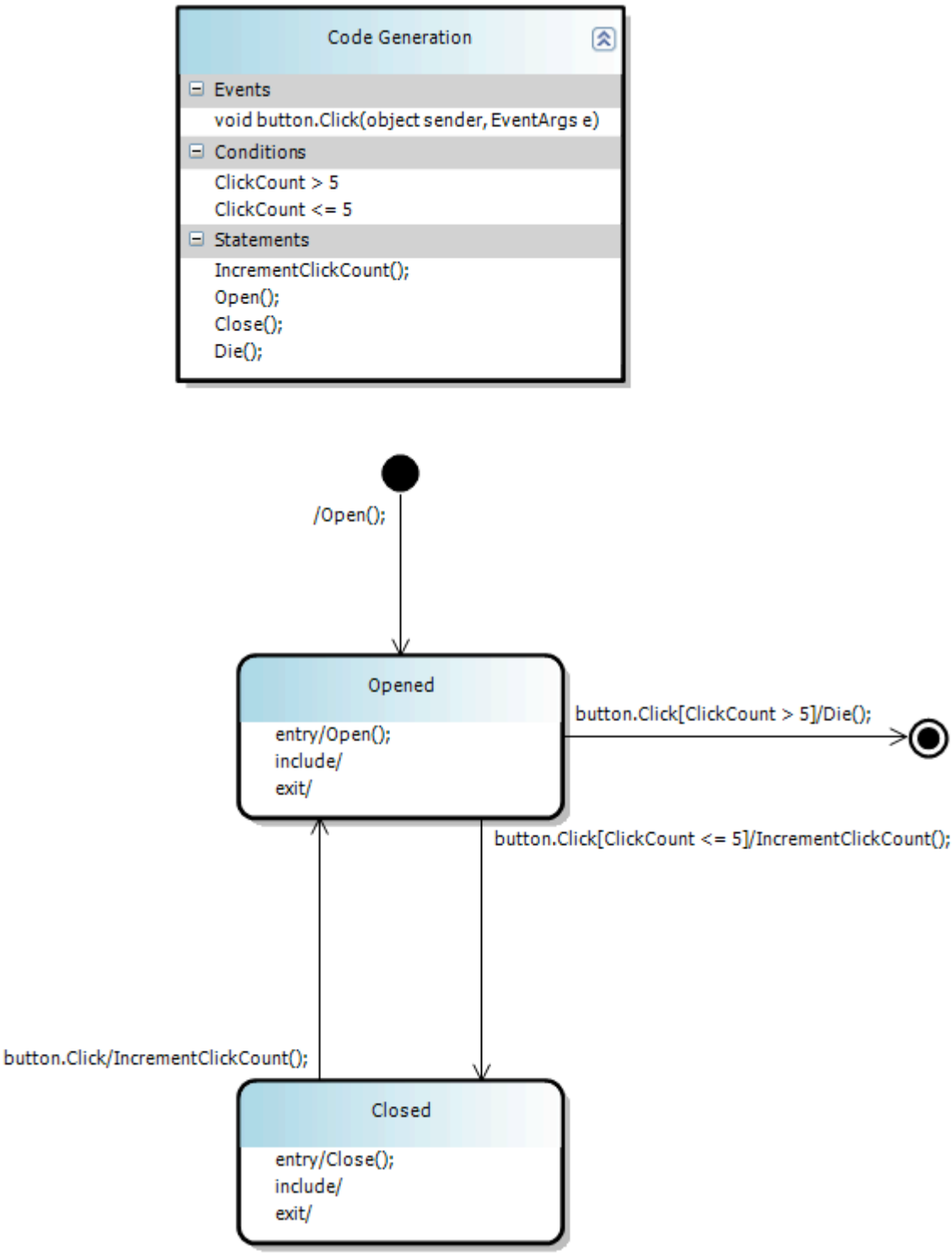


Рис. 12. Диаграмма автомата

Заполняем заглушки методов, созданные редактором классов:

```
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class ButtonClicker
    {
        public ButtonClicker( Button button )
        {
            this.button = button;
        }

        private Button button;
        private int count = 0;

        public int ClickCount
        {
            get {  return count; }
        }

        public void Open()
        {
            button.Text = "Opened";
        }

        public void Close()
        {
            button.Text = "Closed";
        }

        public void IncrementClickCount()
        {
            count++;
        }

        public void Die()
        {
            button.Text = "Dead";
        }
    }
}
```

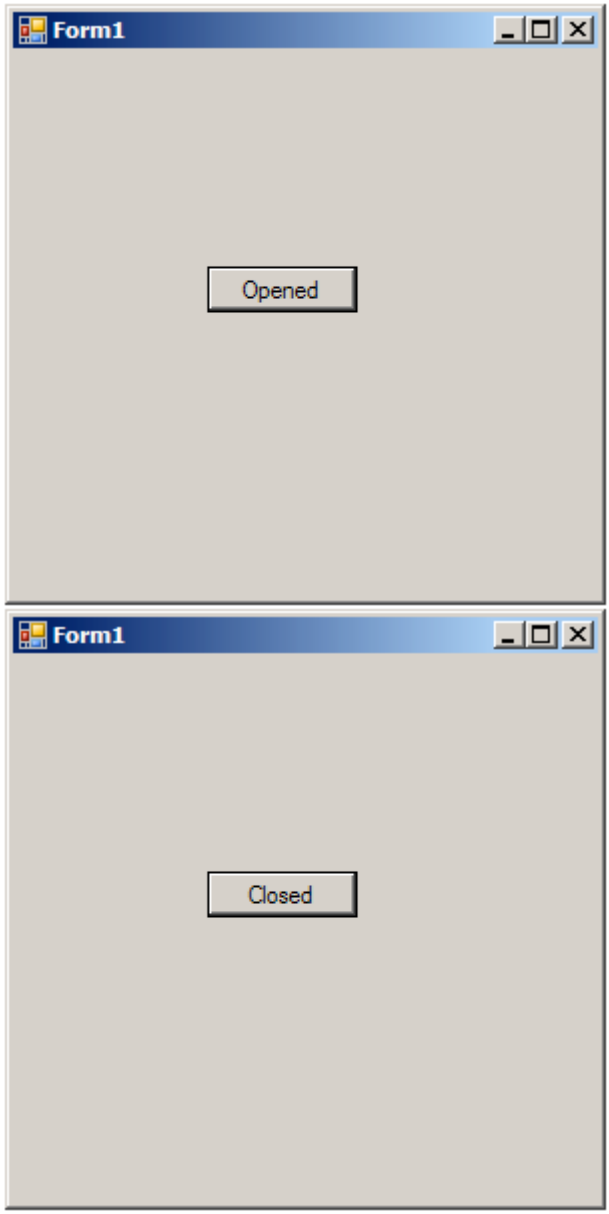
Связываем кнопку на форме с классом `ButtonClicker`.

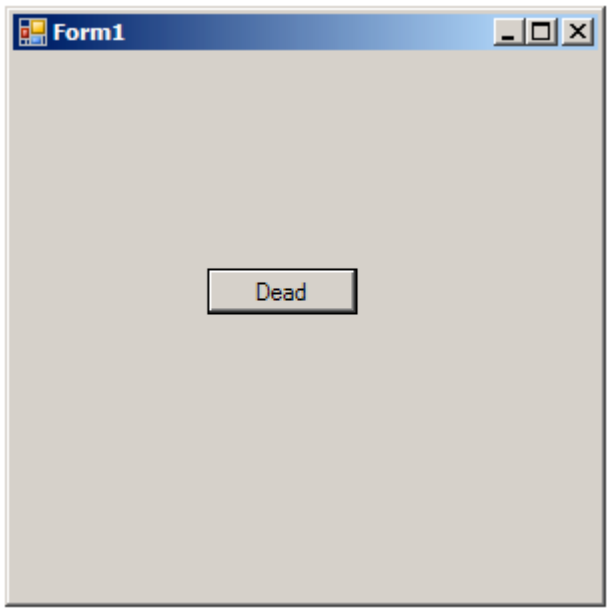
```
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            ButtonClicker buttonTrigger = new ButtonClicker( button1 );
            buttonTrigger.Start();
        }
    }
}
```

Результаты работы приложения:




Form1
Dead

Ниже приведен сгенерированный по модели автомата код:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Threading;

// namespace of StateMachine class
namespace WindowsApplication1
{
    // StateMachine class
    public partial class ButtonClicker
    {
        // Enum of StateMachine's States
        private enum States_9aef7939bd9747dd8641a55053c0981d
        {
            Final,
            Transition,
            Opened_46bfaad81e58454485293f5d8c11720e,
            Closed_d17f2043d2f6433c9f72956efbe575dc,
        }

        // Lock object for StateMachine's data
        private object stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 = new object();

        // Field contains current state of StateMachine
        private States_9aef7939bd9747dd8641a55053c0981d currentState_a7edb23e09f74bd4931323d28e2b3074 =
States_9aef7939bd9747dd8641a55053c0981d.Final;

        // Field contains queue of event threads for sync StateMachine
        private Queue<Thread> eventQueue_2c3199d88dee4d8695e505083a8554ba = new Queue<Thread>();

        public void Start()
        {
            lock ( eventQueue_2c3199d88dee4d8695e505083a8554ba )
            {
                if ( ( 0 < eventQueue_2c3199d88dee4d8695e505083a8554ba.Count ) &&
                    ( Thread.CurrentThread == eventQueue_2c3199d88dee4d8695e505083a8554ba.Peek() ) )
                    throw new InvalidOperationException( "Processed event can't start StateMachine
again!" );

                eventQueue_2c3199d88dee4d8695e505083a8554ba.Enqueue( Thread.CurrentThread );
                while ( Thread.CurrentThread != eventQueue_2c3199d88dee4d8695e505083a8554ba.Peek() )
                    Monitor.Wait( eventQueue_2c3199d88dee4d8695e505083a8554ba );
            }

            try
            {
            lock ( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 )
            {
                System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] StateMachine is starting...", "StateMachineTrace"
);

                if ( States_9aef7939bd9747dd8641a55053c0981d.Final !=
currentState_a7edb23e09f74bd4931323d28e2b3074 )
                    throw new InvalidOperationException( "State Machine is already started!" );

                SubscribeEvents_facdc1274b1547ee95dc247d2085847d();

                OnOpenedEntry_46bfaad81e58454485293f5d8c11720e();
            }
            }
            finally
            {
                lock ( eventQueue_2c3199d88dee4d8695e505083a8554ba )
                {
                    if ( Thread.CurrentThread != eventQueue_2c3199d88dee4d8695e505083a8554ba.Dequeue()
)
                        throw new InvalidOperationException( "Abnormal event processing! This exception
must never been thrown!" );
                    Monitor.PulseAll( eventQueue_2c3199d88dee4d8695e505083a8554ba );
                }
            }
```

```
        }

        public void Join()
        {
            lock ( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 )
            {
                while ( States_9aef7939bd9747dd8641a55053c0981d.Final !=
currentState_a7edb23e09f74bd4931323d28e2b3074 )
                    Monitor.Wait( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 );
            }
        }

        public void Abort()
        {
            lock ( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 )
            {
                if ( States_9aef7939bd9747dd8641a55053c0981d.Final ==
currentState_a7edb23e09f74bd4931323d28e2b3074 )
                    return;

                OnFinalStateEntry_63c28a54e167432a8bf633690f3bf66e();

                Monitor.PulseAll( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 );
            }
        }

        private void OnOpenedEntry_46bfaad81e58454485293f5d8c11720e()
        {
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] entering state Opened...", "StateMachineTrace" );
            currentState_a7edb23e09f74bd4931323d28e2b3074 =
States_9aef7939bd9747dd8641a55053c0981d.Opened_46bfaad81e58454485293f5d8c11720e;
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing entry activity 'Open();'...",
"StateMachineTrace" );
            Open(); // User's provided activity
        }

        private void OnOpenedExit_46bfaad81e58454485293f5d8c11720e()
        {
            currentState_a7edb23e09f74bd4931323d28e2b3074 =
States_9aef7939bd9747dd8641a55053c0981d.Transition;
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] exiting state Opened...", "StateMachineTrace" );
        }

        private void OnClosedEntry_d17f2043d2f6433c9f72956efbe575dc()
        {
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] entering state Closed...", "StateMachineTrace" );
            currentState_a7edb23e09f74bd4931323d28e2b3074 =
States_9aef7939bd9747dd8641a55053c0981d.Closed_d17f2043d2f6433c9f72956efbe575dc;
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing entry activity 'Close();'...",
"StateMachineTrace" );
            Close(); // User's provided activity
        }

        private void OnClosedExit_d17f2043d2f6433c9f72956efbe575dc()
        {
            currentState_a7edb23e09f74bd4931323d28e2b3074 =
States_9aef7939bd9747dd8641a55053c0981d.Transition;
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] exiting state Closed...", "StateMachineTrace" );
        }

        private void OnFinalStateEntry_63c28a54e167432a8bf633690f3bf66e()
        {
            currentState_a7edb23e09f74bd4931323d28e2b3074 =
States_9aef7939bd9747dd8641a55053c0981d.Final;
            UnsubscribeEvents_a438d4a162fc455c91dae84cb363f23b();


            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] StateMachine is stopping...", "StateMachineTrace"
);
        }

        private void HandlebuttonClick_dc6ac9b499944822ba39742a9ccd96df( object sender, EventArgs e )
        {
```

```
            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] processing event button.Click...",
"StateMachineTrace" );

            lock ( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 )
            {
                switch ( currentState_a7edb23e09f74bd4931323d28e2b3074 )
                {
                    case
States_9aef7939bd9747dd8641a55053c0981d.Opened_46bfaad81e58454485293f5d8c11720e:
                        if ( ClickCount <= 5 )
                        {
                            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] guard 'ClickCount <= 5' is true...",
"StateMachineTrace" );

                            OnOpenedExit_46bfaad81e58454485293f5d8c11720e();
                            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity
'IncrementClickCount();'...", "StateMachineTrace" );
                            IncrementClickCount(); // User's provided activity
                            OnClosedEntry_d17f2043d2f6433c9f72956efbe575dc();
                        }
                        else if ( ClickCount > 5 )
                        {
                            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] guard 'ClickCount > 5' is true...",
"StateMachineTrace" );

                            OnOpenedExit_46bfaad81e58454485293f5d8c11720e();
                            System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity 'Die();'...",
"StateMachineTrace" );
                            Die(); // User's provided activity
                            OnFinalStateEntry_63c28a54e167432a8bf633690f3bf66e();
                        }
                        break;
                    case
States_9aef7939bd9747dd8641a55053c0981d.Closed_d17f2043d2f6433c9f72956efbe575dc:
                        OnClosedExit_d17f2043d2f6433c9f72956efbe575dc();
                        System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity
'IncrementClickCount();'...", "StateMachineTrace" );
                        IncrementClickCount(); // User's provided activity
                        OnOpenedEntry_46bfaad81e58454485293f5d8c11720e();
                        break;
                    default:
                        System.Diagnostics.Trace.WriteLine( "ButtonClicker [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] StateMachine button.Click has not being
processed...", "StateMachineTrace" );
                        break;
                }

                Monitor.PulseAll( stateMachineLockObject_16e9ec5b5e304444945696d8ecb7d784 );
            }
        }


        private void OnbuttonClick_dc6ac9b499944822ba39742a9ccd96df( object sender, EventArgs e )
        {
            lock ( eventQueue_2c3199d88dee4d8695e505083a8554ba )
            {
                if ( ( 0 < eventQueue_2c3199d88dee4d8695e505083a8554ba.Count ) &&
                    ( Thread.CurrentThread == eventQueue_2c3199d88dee4d8695e505083a8554ba.Peek() ) )
                    throw new InvalidOperationException( "Processed event can't produce events!" );

                eventQueue_2c3199d88dee4d8695e505083a8554ba.Enqueue( Thread.CurrentThread );
                while ( Thread.CurrentThread != eventQueue_2c3199d88dee4d8695e505083a8554ba.Peek() )
                    Monitor.Wait( eventQueue_2c3199d88dee4d8695e505083a8554ba );
            }

            try
            {
                HandlebuttonClick_dc6ac9b499944822ba39742a9ccd96df( sender, e );
            }
            finally
            {
                lock ( eventQueue_2c3199d88dee4d8695e505083a8554ba )
                {
                    if ( Thread.CurrentThread != eventQueue_2c3199d88dee4d8695e505083a8554ba.Dequeue()
)
```

```
                        throw new InvalidOperationException( "Abnormal event processing! This exception
must never been thrown!" );
                    Monitor.PulseAll( eventQueue_2c3199d88dee4d8695e505083a8554ba );
                }
            }
        }

        private void SubscribeEvents_facdc1274b1547ee95dc247d2085847d()
        {
            button.Click += OnbuttonClick_dc6ac9b499944822ba39742a9ccd96df;
        }

        private void UnsubscribeEvents_a438d4a162fc455c91dae84cb363f23b()
        {
            button.Click -= OnbuttonClick_dc6ac9b499944822ba39742a9ccd96df;
        }
    }
}
```

5.2. Обедающие философы

Данный пример показывает возможности использования инструментального средства для решения многопоточных задач. В нем приведено решение задачи синхронизации нескольких потоков – задачи обедающих философов [11]. Задачи обедающих философов является наглядным примером задачи параллельного вычисления. Это классическая задача синхронизации нескольких процессов. В 1971 году Эдсгер Дейкстра поставил вопрос о проблеме синхронизации пяти компьютеров конкурирующих за доступ к пяти периферийным устройствам записи на ленту. Вскоре после этого Тони Хоар переформулировал задачу как задачу об обедающих философах.

Пять философов проводят время, думая и обедая. Они сидят за столом, в центре которого стоит большое блюдо со спагетти. Перед каждым философом стоит тарелка, а между каждой соседней тарелкой лежит по одной вилке (рис. 13).

Для того чтобы есть спагетти философу необходимо использовать две вилки, которые философы берут по одной. Философы не разговаривают друг с другом, что создает возможность взаимоблокировки, при которой у каждого философа взята одна вилка, левая или правая, и он ждет освобождения другой.

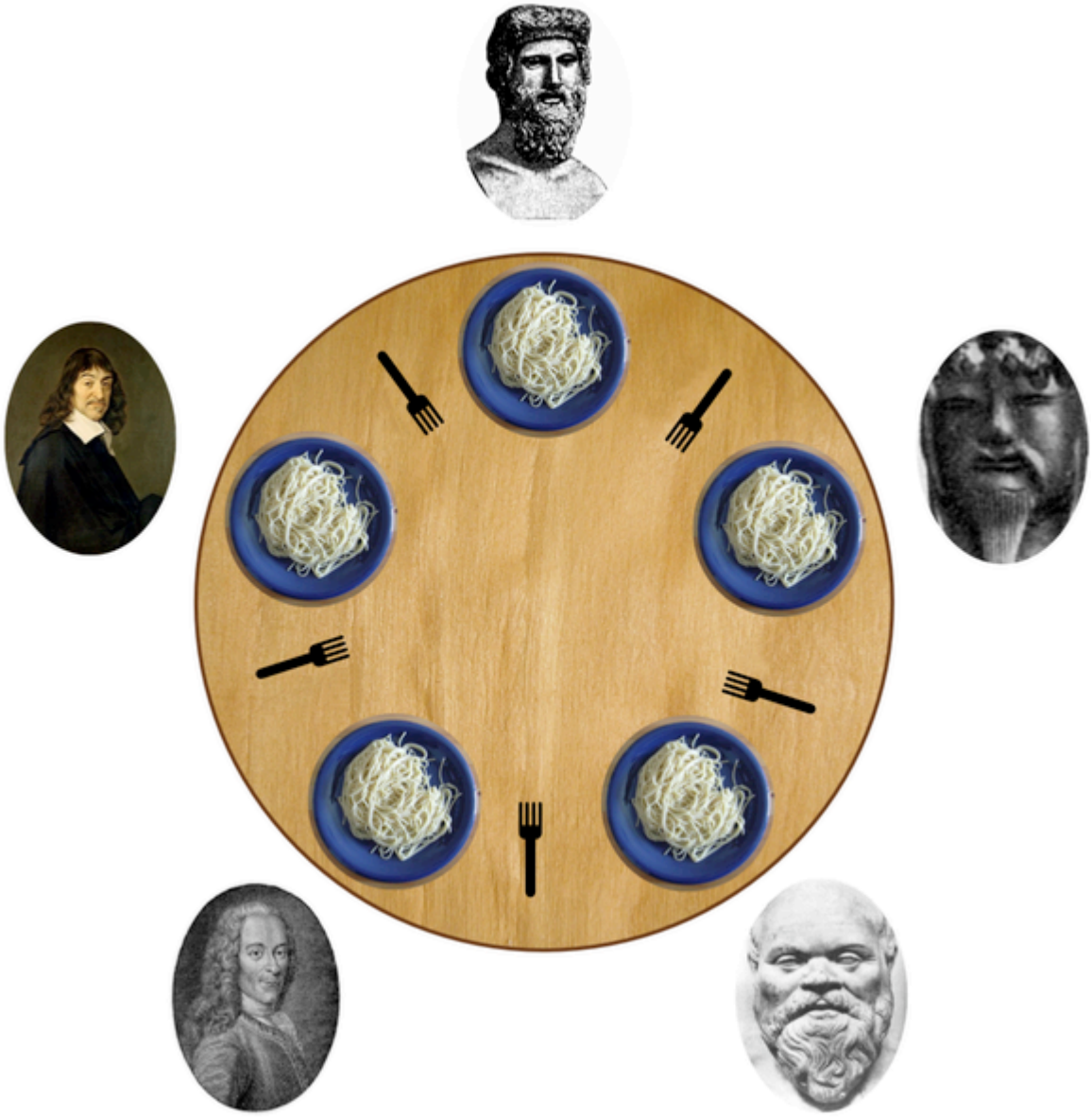


Рис. 13. Иллюстрация к задаче обедающих философов

Дейкстра предложил следующее решение этой задачи. Все философы, кроме одного, берут вилки в одном порядке, например, сначала левую, а потом правую, а один выделенный философ ведет себя наоборот, то есть берет сначала правую, а потом левую. Таким образом, добавляется асимметрия, что предотвращает возникновение взаимоблокировки.

В данном примере используется адаптация для автоматной модели решения, которое предложили в 1984 году К. М. Чанди и Дж. Мисра. Решение заключается в том, что вилки маркируются как грязные или чистые и философы запрашивают вилки у соседей. Если запрашиваемая вилка чистая, то философ не отдает ее. При

передаче вилки маркируются как чистые, а после использования маркируются как грязные. Изначально вилки раздаются философам следующим образом: первый получает обе вилки, последующие по одной, а последний остается без вилок.

В данном примере приведено два автоматных решения. В первом случаи используется автомат класса `Philosopher` с 15 состояниями, а во втором с тремя. В обоих случаях исполняется система из пяти взаимодействующих автоматов одного класса `Philosopher`, взаимно использующих пять объектов класса `Fork`.

На рис. 14 приведена диаграмма классов для решения с 15 состояниями.

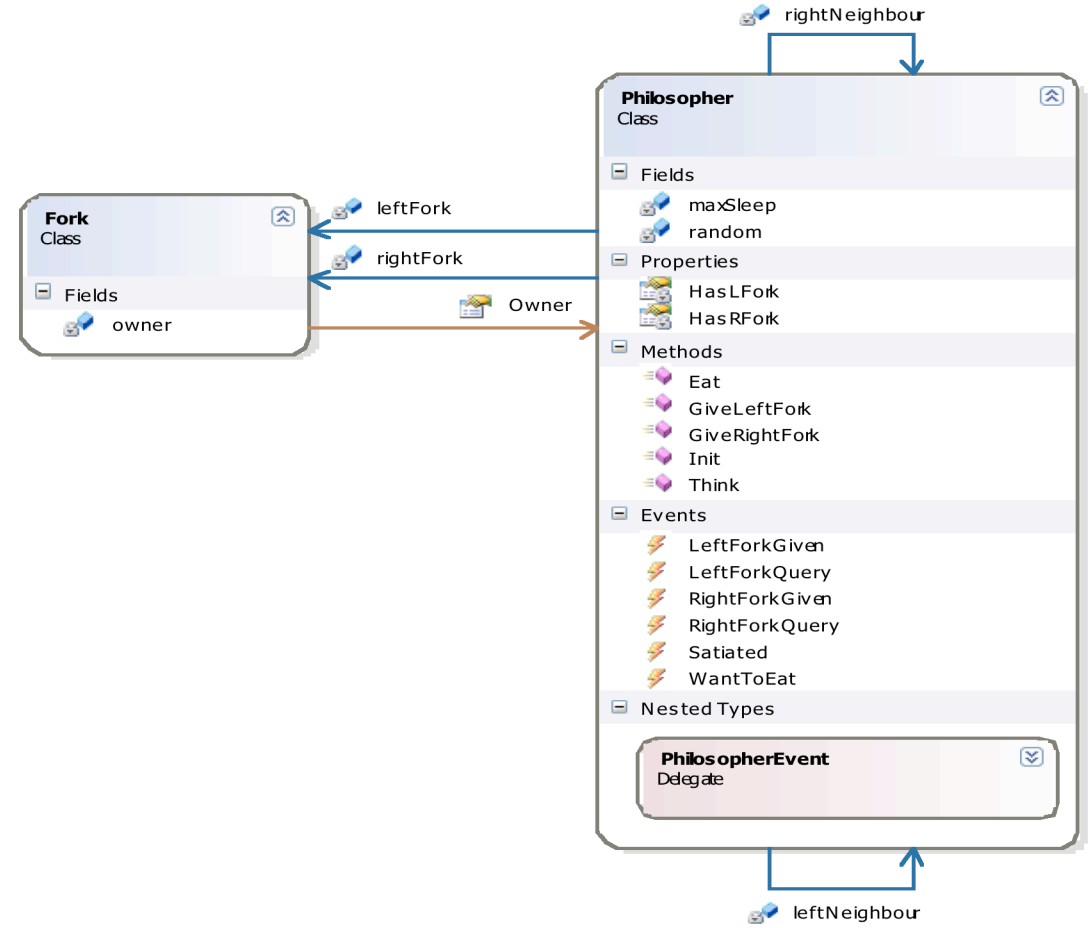


Рис. 14. Диаграмма классов решения задачи обедающих философов с 15 состояниями

Ниже приведены исходные тексты главного метода программы, класса `Philosopher` и класса `Fork` решения с 15 состояниями.

Главный метод:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;

namespace DiningPhilosophersSample
{
    internal class Program
    {
        private const uint philosophersCount = 5;

        private static Dictionary<Philosopher, int> eatTimes = new Dictionary<Philosopher, int>();

        private static void Main( string[] args )
        {
            Philosopher[] philosophers = new Philosopher[ philosophersCount ];
            Fork[] forks = new Fork[ philosophersCount ];
            for ( int i = 0; i < philosophers.Length; i++ )
                philosophers[ i ] = new Philosopher();
            for ( int i = 0; i < forks.Length; i++ )
                forks[ i ] = new Fork();

            for ( int i = 0; i < philosophers.Length; i++ )
            {
                eatTimes.Add( philosophers[ i ], 0 );
                philosophers[ i ].Satiated += new EventHandler( Program_Satiated );

                philosophers[ i ].Init(
                    philosophers[ ( philosophersCount + i - 1 ) % philosophersCount ],
                    philosophers[ ( i + 1 ) % philosophersCount ],
                    forks[ i ],
                    forks[ ( i + 1 ) % philosophersCount ],
                    0 == i,
                    philosophersCount - 1 != i );
            }

            foreach ( Philosopher philosopher in philosophers )
                philosopher.Init();

            foreach ( Philosopher philosopher in philosophers )
                philosopher.Start();
        }

        private static void Program_Satiated( object sender, EventArgs e )
        {
            Philosopher philosopher = sender as Philosopher;
            if ( null == philosopher )
                return;

            eatTimes[ philosopher ] = eatTimes[ philosopher ] + 1;

            StringBuilder stringBuilder = new StringBuilder( "Philosophers eat times: " );
            Philosopher[] philosophers = new Philosopher[ eatTimes.Keys.Count ];
            eatTimes.Keys.CopyTo( philosophers, 0 );
            for ( int i = 0; i < philosophers.Length; i++ )
                stringBuilder.Append( eatTimes[ philosophers[ i ] ].ToString( "{0} " ) );

            Trace.WriteLine( stringBuilder.ToString() );
        }
    }
}
```

Класс `Philosopher`:

```
using System;
using System.Threading;

namespace DiningPhilosophersSample
{
    public partial class Philosopher
    {
        public void Init( Philosopher leftNeighbour, Philosopher rightNeighbour,
                          Fork leftFork, Fork rightFork,
                          bool takeLeftFork, bool takeRightFork )
        {
            if ( null == leftNeighbour )
                throw new ArgumentNullException( "leftNeighbour" );
            if ( null == rightNeighbour )
                throw new ArgumentNullException( "rightNeighbour" );
            this.leftNeighbour = leftNeighbour;
            this.rightNeighbour = rightNeighbour;

            if ( null == leftFork )
                throw new ArgumentNullException( "leftFork" );
            if ( null == rightFork )
                throw new ArgumentNullException( "rightFork" );
            this.leftFork = leftFork;
            this.rightFork = rightFork;

            if ( takeLeftFork )
                this.leftFork.Owner = this;
            if ( takeRightFork )
                this.rightFork.Owner = this;
        }

        private Philosopher leftNeighbour;
        private Philosopher rightNeighbour;

        private Fork leftFork;
        private Fork rightFork;

        private bool HasLFork
        {
            get { return ( this == leftFork.Owner ); }
        }

        private bool HasRFork
        {
            get { return ( this == rightFork.Owner ); }
        }

        public delegate void EmptyEvent();

        public delegate void PhilosopherEvent( Philosopher philosopher );

        public event PhilosopherEvent WantToEat;

        public event PhilosopherEvent Satiated;

        public event EmptyEvent LeftForkQuery;

        public event EmptyEvent RightForkQuery;

        public event EmptyEvent LeftForkGiven;

        public event EmptyEvent RightForkGiven;

        private const int maxSleep = 1000;

        private Random random = new Random();

        public void Think()
        {
            Thread.Sleep( random.Next( maxSleep ) );

            WantToEat( this );
        }

        public void GiveLeftFork()
        {
            if ( !HasLFork )
                throw new InvalidOperationException( "Philosopher has not LFork!" );
```

```
            if ( leftNeighbour == leftFork.Owner )
                throw new InvalidOperationException( "Left Neighbour already has RFork!" );

            leftFork.Owner = leftNeighbour;

            LeftForkGiven();
        }


        public void GiveRightFork()
        {
            if ( !HasRFork )
                throw new InvalidOperationException( "Philosopher has not RFork!" );

            if ( rightNeighbour == rightFork.Owner )
                throw new InvalidOperationException( "Right Neighbour already has LFork!" );

            rightFork.Owner = rightNeighbour;

            RightForkGiven();
        }

        public void Eat()
        {
            if ( !HasLFork && !HasRFork )
                throw new InvalidOperationException( "Philosopher cannot eat without Forks!" );
            if ( !HasLFork )
                throw new InvalidOperationException( "Philosopher cannot eat without LFork!" );
            if ( !HasRFork )
                throw new InvalidOperationException( "Philosopher cannot eat without RFork!" );

            Thread.Sleep( random.Next( maxSleep ) );

            Satiated( this );
        }
    }
}
```

Класс `Fork`:

```
namespace DiningPhilosophersSample
{
    public class Fork
    {
        private Philosopher owner = null;

        public Philosopher Owner
        {
            get { return owner; }
            set { owner = value; }
        }
    }
}
```

На рис. 15 приведена диаграмма состояний объекта класса `Philosopher` для решения с 15 состояниями.

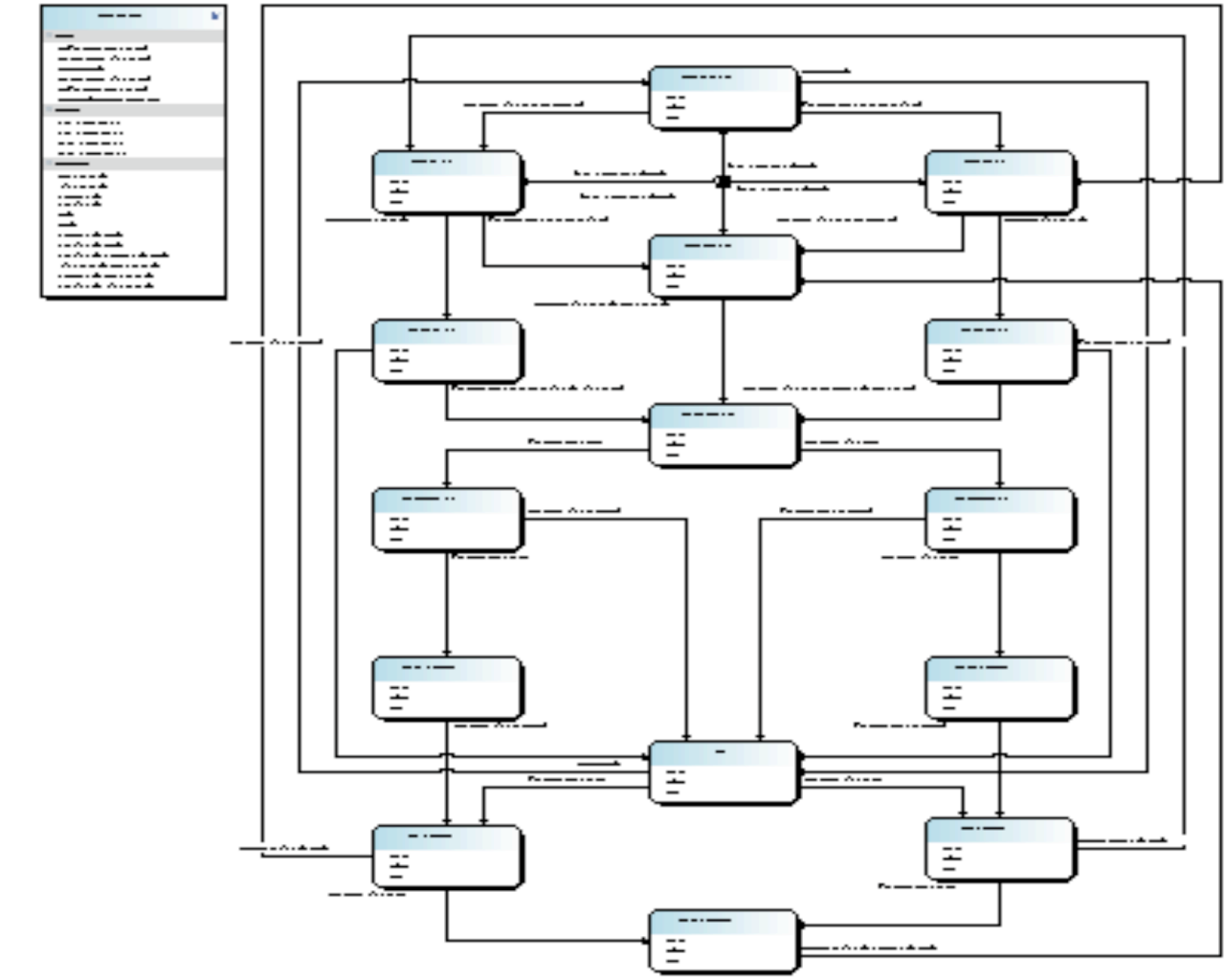


Рис. 15. Автомат класса `Philosopher` программы решения задачи обедающих философов с 15 состояниями

При решении с использованием автомата из трех состояний для класса `Philosopher` приходится заводить дополнительные методы и поля. Например, поле `clear` и свойство `Clear` в классе `Fork` для возможности узнать чистая ли вилка или нет, поле `forksToGive` и метод `GiveQueriedForks` в классе `Philosopher` для запоминания и передачи всех запрошенных вилок. А также

нужно добавить дополнительные параметры в события о запросе и передаче вилок (рис. 16).

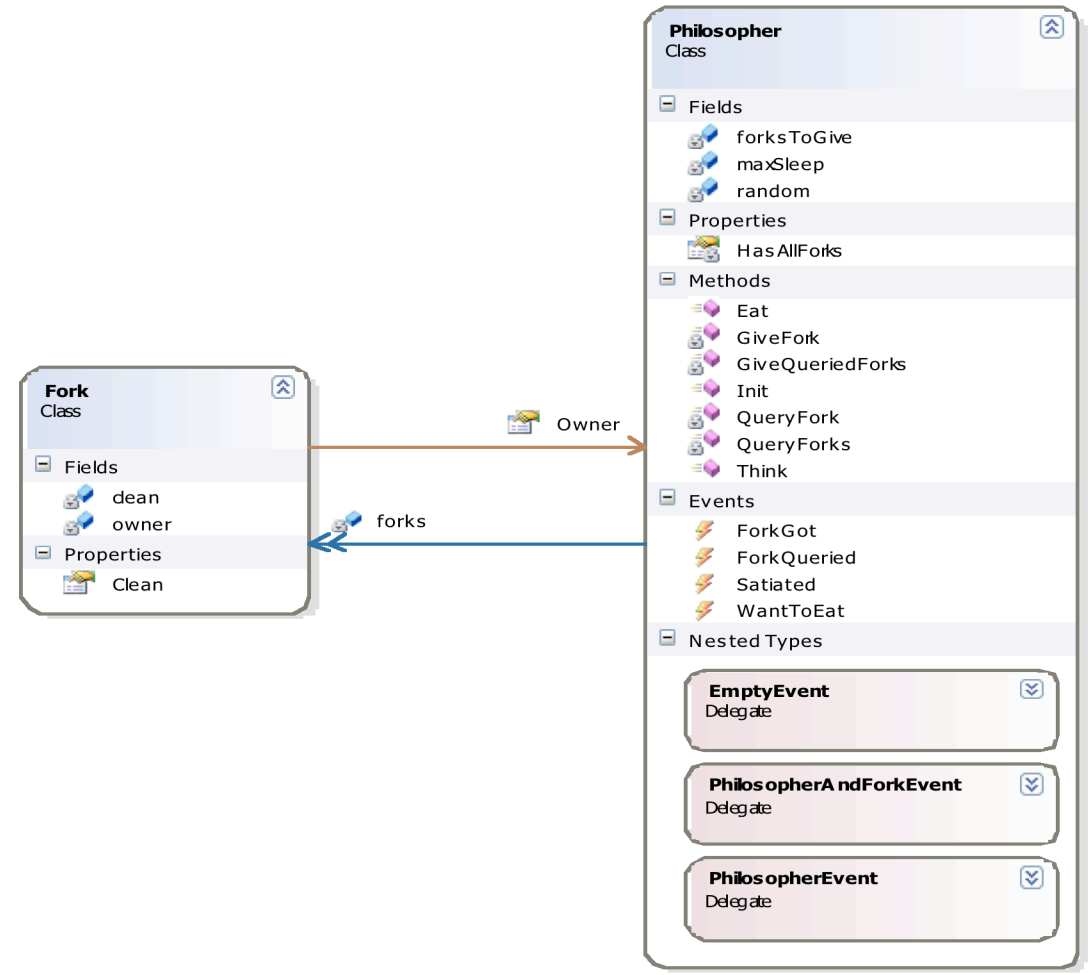


Рис. 16. Диаграмма классов решения задачи обедающих философов с тремя состояниями

На рис. 17 приведена диаграмма состояний объекта класса `Philosopher` для решения с тремя состояниями.

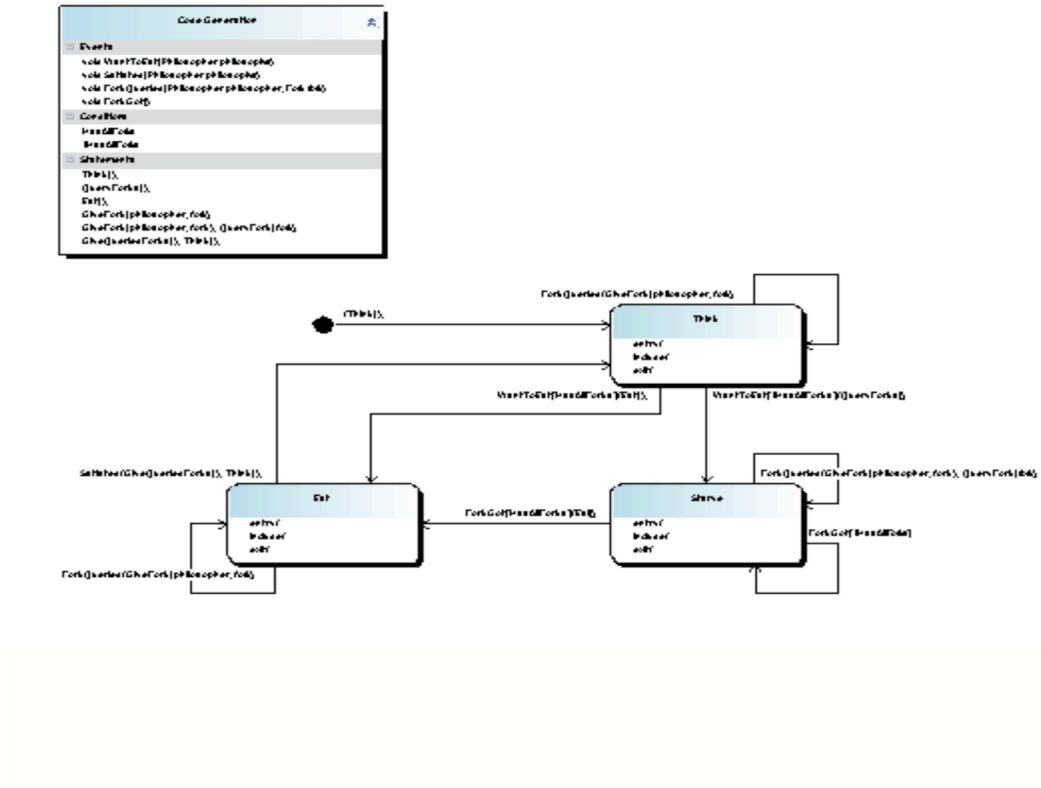


Рис. 17. Автомат класса `Philosopher` программы решения задачи обедающих философов с тремя состояниями

Ниже приведены исходные коды классов *Fork* и *Philosopher* для решении с тремя состояниями, где часть логики работы по синхронизации внесена код.

Класс `Philosopher`:

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace DiningPhilosophersSample
{
    public partial class Philosopher
    {
        public void Init(   Fork    leftFork,       Fork    rightFork,
                            bool    takeLeftFork,   bool    takeRightFork )
        {
            if ( null == leftFork )
                throw new ArgumentNullException( "leftFork" );
            if ( null == rightFork )
                throw new ArgumentNullException( "rightFork" );

            forks.Add( leftFork );
            forks.Add( rightFork );

            if ( takeLeftFork )
                leftFork.Owner = this;
            if ( takeRightFork )
                rightFork.Owner = this;
        }

        private List<Fork> forks = new List<Fork>();

        private List<KeyValuePair<Philosopher, Fork>> forksToGive = new List<KeyValuePair<Philosopher,
Fork>>();
```

```
private bool HasAllForks
{
    get
    {
        foreach ( Fork fork in forks )
        {
            if ( this != fork.Owner )
                return false;
        }

        return true;
    }
}

private void QueryForks()
{
    foreach ( Fork fork in forks )
        QueryFork( fork );
}

private void GiveFork( Philosopher philosopher, Fork fork )
{
    if ( this != fork.Owner )
        throw new InvalidOperationException( "Philosopher has not such fork!" );

    if ( fork.Clean )
    {
        forksToGive.Add( new KeyValuePair<Philosopher,Fork>( philosopher, fork ) );
    }
    else
    {
        fork.Clean = true;
        fork.Owner = philosopher;
        philosopher.ForkGot();
    }
}

private void QueryFork( Fork fork )
{
    if ( this != fork.Owner )
        fork.Owner.ForkQueried( this, fork );
}

private void GiveQueriedForks()
{
    foreach ( KeyValuePair<Philosopher, Fork> philosopherAndFork in forksToGive )
        GiveFork( philosopherAndFork.Key, philosopherAndFork.Value );

    forksToGive.Clear();
}


public delegate void EmptyEvent();

public delegate void PhilosopherEvent( Philosopher philosopher );

public delegate void PhilosopherAndForkEvent( Philosopher philosopher, Fork fork );

public event PhilosopherEvent WantToEat;

public event PhilosopherEvent Satiated;

public event PhilosopherAndForkEvent ForkQueried;

public event EmptyEvent ForkGot;

private const int maxSleep = 1000;

private Random random = new Random();

public void Think()
{
    Thread.Sleep( random.Next( maxSleep ) );

    WantToEat( this );
}

public void Eat()
{
    if ( !HasAllForks )
        throw new InvalidOperationException( "Philosopher cannot eat without Forks!" );
```

```
            Thread.Sleep( random.Next( maxSleep ) );

            foreach ( Fork fork in forks )
            {
                fork.Clean = false;
            }

            Satiated( this );
        }
    }
}
```

Класс `Fork`:

```
namespace DiningPhilosophersSample
{
    public class Fork
    {
        private Philosopher owner = null;

        public Philosopher Owner
        {
            get { return owner; }
            set { owner = value; }
        }

        private bool clean = false;

        public bool Clean
        {
            get { return clean; }
            set { clean = value; }
        }
    }
}
```

ЗАКЛЮЧЕНИЕ

Результат выполненной работы – инструментальное средство визуального проектирования автоматов. Дополняя статическую модель разрабатываемых программ визуально представленную на *Диаграмме Классов* среды разработки *Microsoft Visual Studio 2005*, визуальный язык автоматного программирования позволяет разрабатывать и реализовывать динамическую логику выполнения программ.

Инструментальное средство может быть использовано при решении широкого класса задач построения автоматных систем. Эта возможность достигается путем удобства использования в качестве различных свойств модели автомата артефактов целевого языка. Это позволяет связывать модель автомата с любым уже имеющимся исходным кодом.

ИСТОЧНИКИ

1. *The Object Management Group (OMG).* OMG Model Driven Architecture. http://www.omg.org/mda/
2. *The Object Management Group (OMG).* UML 2.0 Superstructure specification. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML.
3. *Microsoft Corporation.* Microsoft Visual Studio 2005. http://msdn.microsoft.com/vstudio/.
4. *Троелсен Э.* C# и платформа .Net. Библиотека программиста. СПб.: Питер, 2003.
5. *Тай Т, Лэм Х. К.* Платформа .Net. Основы. O'Reilly. СПб.: Символ, 2003.
6. *Microsoft Visual Studio Developer Center.* Domain-Specific Language Tools. http://msdn.microsoft.com/vstudio/DSLTools/.
7. *Фаулер М.* UML. Основы. Краткое руководство по стандартному языку объектного моделирования. СПб.: Символ, 2005.
8. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
9. *Мартин Р.* Designing Object-Oriented C++ Applications Using The Booch Method. NJ: Prentice Hall, 1993.
10. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. МЦНМО, 2000.
11. *Wikipedia, the free encyclopedia.* Dining philosophers problem. http://en.wikipedia.org/wiki/Dining_philosophers_problem.

ПРИЛОЖЕНИЕ. Шаблон генерации кода

```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation"#>
<#@ output extension=".statemachine.cs" #>
<#@ statemachine processor="StateMachineLanguageDirectiveProcessor"
requires="fileName='%MODELFILENAME%'" provides="StateMachine=StateMachine" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Text.RegularExpressions" #>
<#+
private static string MakeSingleWord( string name, string replacer )
{
    return Regex.Replace( name, "[^A-Za-z_0-9]", replacer );
}

private Dictionary<string, string> uniqueNames = new Dictionary<string, string>();

private string GetUniqueName( string name )
{
    if ( !uniqueNames.ContainsKey( name ) )
        uniqueNames.Add( name, MakeSingleWord( name, string.Empty ) + "_" + Guid.NewGuid().ToString(
"N" ) );

    return uniqueNames[ name ];
}

// *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

private List< KeyValuePair< string, string > > ParseParameters( string parameters )
{
    List< KeyValuePair< string, string > > parsedParameters = new List< KeyValuePair< string, string >
>();

    if ( string.Empty != parameters.Trim() )
    {
        string[] splittedParameters = parameters.Split( new Char[] { ',' } );
        foreach ( string parameter in splittedParameters )
        {
            string[] typeAndName = parameter.Split( new Char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries );
            parsedParameters.Add( new KeyValuePair< string, string >( typeAndName[ 0 ], typeAndName[ 1
] ) );
        }
    }

    return parsedParameters;
}

// *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

private List<Transition> GetOutTransitions( BaseState baseState )
{
    List<Transition> outTransitions = new List<Transition>();

    foreach ( Transition transition in baseState.GetElementLinks( Transition.TransitionFromMetaRoleGuid
) )
        outTransitions.Add( transition );

    return outTransitions;
}

// *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

private Dictionary<BaseState, List<Transition>> GroupTransitionsByState( Event oEvent )
{
    Dictionary<BaseState, List<Transition>> transitionsByState = new Dictionary<BaseState,
List<Transition>>();

    foreach ( Transition transition in oEvent.Transitions )
    {
        if ( !transitionsByState.ContainsKey( transition.TransitionFrom ) )
            transitionsByState.Add( transition.TransitionFrom, new List<Transition>() );

        transitionsByState[ transition.TransitionFrom ].Add( transition );
    }

    return transitionsByState;
}
```

```
// *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

private Dictionary<Event, List<Transition>> GroupTransitionsByEvent( BaseState baseState )
{
    Dictionary<Event, List<Transition>> transitionsByEvent = new Dictionary<Event, List<Transition>>();

    foreach ( Transition transition in baseState.GetElementLinks( Transition.TransitionFromMetaRoleGuid
) )
    {
        if ( !transitionsByEvent.ContainsKey( transition.Event ) )
            transitionsByEvent.Add( transition.Event, new List<Transition>() );

        transitionsByEvent[ transition.Event ].Add( transition );
    }

    return transitionsByEvent;
}

// *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

private class StateNames
{
    public string StateName;

    public string EntryMethod;
    public string ExitMethod;

    public StateNames()
    {
    }

    public StateNames( BaseState baseState )
    {
        string alfaNumericName = MakeSingleWord( baseState.Name, string.Empty );
        string alfaNumericGuid = "_" + Guid.NewGuid().ToString( "N" );
        StateName = alfaNumericName + alfaNumericGuid;
        EntryMethod = "On" + alfaNumericName + "Entry" + alfaNumericGuid;
        ExitMethod = "On" + alfaNumericName + "Exit" + alfaNumericGuid;
    }
}

private StateNames finalStateNames = null;

private StateNames GetFinalStateNames()
{
    if ( null == finalStateNames )
    {
        finalStateNames = new StateNames();
        finalStateNames.EntryMethod = "OnFinalStateEntry_" + Guid.NewGuid().ToString( "N" );
    }

    return finalStateNames;
}

private Dictionary<BaseState, StateNames> statesNames = new Dictionary<BaseState, StateNames>();

private StateNames GetStateNames( BaseState baseState )
{
    if ( baseState is FinalState )
        return GetFinalStateNames();

    if ( !statesNames.ContainsKey( baseState ) )
        statesNames.Add( baseState, new StateNames( baseState ) );

    return statesNames[ baseState ];
}

// *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

private class EventNames
{
    public string EventName;

    public string EventSubscribeMethod;

    public string EventHandlerMethod;

    public EventNames( Event oEvent )
    {
```

```
        EventName = MakeSingleWord( oEvent.Access, string.Empty ) + "_" + Guid.NewGuid().ToString( "N"
);
        EventSubscribeMethod = "On" + EventName;
        EventHandlerMethod = "Handle" + EventName;
    }
}

private Dictionary<Event, EventNames> eventNames = new Dictionary<Event, EventNames>();

private EventNames GetEventNames( Event oEvent )
{
    if ( !eventNames.ContainsKey( oEvent ) )
        eventNames.Add( oEvent, new EventNames( oEvent ) );

    return eventNames[  oEvent ];
}
#>
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Threading;

// namespace of StateMachine class
namespace <#= this.StateMachine.CodeGeneration.Namespace #>
{
    // StateMachine class
    <#= this.StateMachine.CodeGeneration.ClassModifiers #> class <#=
this.StateMachine.CodeGeneration.Class #>
    {
        // Enum of StateMachine's States
        private enum <#= GetUniqueName( "States" ) #>
        {
            Final,
<#
    if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>
            Initialized,
<#
    }
#>
            Transition,
<#
    foreach ( BaseState baseState in this.StateMachine.States )
    {
        if ( baseState is State )
        {
#>
            <#= GetStateNames( baseState ).StateName #>,
<#
        }
    }
#>
        }

        // Lock object for StateMachine's data
        private object <#= GetUniqueName( "stateMachineLockObject" ) #> = new object();

        // Field contains current state of StateMachine
        private <#= GetUniqueName( "States" ) #> <#= GetUniqueName( "currentState" ) #> = <#=
GetUniqueName( "States" ) #>.Final;
<#
    if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>

        // Field contains thread for async StateMachine
        private Thread <#= GetUniqueName( "stateMachineThread" ) #> = null;

        // Init method for async StateMachine
        <#= this.StateMachine.CodeGeneration.InitMethodModifiers #> void <#=
this.StateMachine.CodeGeneration.InitMethod #>()
        {
            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
```

```
            {
                if ( <#= GetUniqueName( "States" ) #>.Final != <#= GetUniqueName( "currentState" ) #> )
                    throw new InvalidOperationException( "State Machine is already initialized!" );

                <#= GetUniqueName( "SubscribeEvents" ) #>();

                <#= GetUniqueName( "currentState" ) #> = <#= GetUniqueName( "States" ) #>.Initialized;
            }
        }
<#
    }
    else
    {
#>

        // Field contains queue of event threads for sync StateMachine
        private Queue<Thread> <#= GetUniqueName( "eventQueue" ) #> = new Queue<Thread>();
<#
    }
#>

        <#= this.StateMachine.CodeGeneration.StartMethodModifiers #> void <#=
this.StateMachine.CodeGeneration.StartMethod #>()
        {
<#
    if ( Synchronization.Sync_NoThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>
            lock ( <#= GetUniqueName( "eventQueue" ) #> )
            {
                if ( ( 0 < <#= GetUniqueName( "eventQueue" ) #>.Count ) &&
                    ( Thread.CurrentThread == <#= GetUniqueName( "eventQueue" ) #>.Peek() ) )
                    throw new InvalidOperationException( "Processed event can't start StateMachine
again!" );

                <#= GetUniqueName( "eventQueue" ) #>.Enqueue( Thread.CurrentThread );
                while ( Thread.CurrentThread != <#= GetUniqueName( "eventQueue" ) #>.Peek() )
                    Monitor.Wait( <#= GetUniqueName( "eventQueue" ) #> );
            }

            try
            {
<#
    }
#>
            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
            {
                System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "]  StateMachine is starting...", "StateMachineTrace"
);

<#
    if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>
                if ( <#= GetUniqueName( "States" ) #>.Final == <#= GetUniqueName( "currentState" ) #> )
                    <#= this.StateMachine.CodeGeneration.InitMethod #>();

                if ( <#= GetUniqueName( "States" ) #>.Initialized != <#= GetUniqueName( "currentState"
) #> )
                    throw new InvalidOperationException( "State Machine is already started!" );
<#
    }
    else
    {
#>
                if ( <#= GetUniqueName( "States" ) #>.Final != <#= GetUniqueName( "currentState" ) #> )
                    throw new InvalidOperationException( "State Machine is already started!" );

                <#= GetUniqueName( "SubscribeEvents" ) #>();
<#
    }
#>

<#
    List<Transition> startTransitions = GetOutTransitions( this.StateMachine.StartState );
    IEnumerator<Transition> transitionEnumerator = startTransitions.GetEnumerator();
    transitionEnumerator.MoveNext();

    if ( "" != transitionEnumerator.Current.Guard )
    {
```

```
#>
                if ( <#= transitionEnumerator.Current.Guard #> )
                {
                    System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #>
[ " + Thread.CurrentThread.ManagedThreadId.ToString() + "] guard '<#= transitionEnumerator.Current.Guard
#>' is true...", "StateMachineTrace" );

<#
        if ( "" != transitionEnumerator.Current.Activity )
        {
#>
                    System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #>
[ " + Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity '<#=
transitionEnumerator.Current.Activity #>'...", "StateMachineTrace" );
                    <#= transitionEnumerator.Current.Activity #> // User's provided activity
<#
        }
#>
                    <#= GetStateNames( transitionEnumerator.Current.TransitionTo ).EntryMethod #>();
                }
<#
        while ( transitionEnumerator.MoveNext() )
        {
#>
                else if ( <#= transitionEnumerator.Current.Guard #> )
                {
                    System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #>
[ " + Thread.CurrentThread.ManagedThreadId.ToString() + "] guard '<#= transitionEnumerator.Current.Guard
#>' is true...", "StateMachineTrace" );

<#
            if ( "" != transitionEnumerator.Current.Activity )
            {
#>
                    System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #>
[ " + Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity '<#=
transitionEnumerator.Current.Activity #>'...", "StateMachineTrace" );
                    <#= transitionEnumerator.Current.Activity #> // User's provided activity
<#
            }
#>
                    <#= GetStateNames( transitionEnumerator.Current.TransitionTo ).EntryMethod #>();
                }
<#
        }
    }
    else
    {
        if ( "" != transitionEnumerator.Current.Activity )
        {
#>
                System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity '<#=
transitionEnumerator.Current.Activity #>'...", "StateMachineTrace" );
                <#= transitionEnumerator.Current.Activity #> // User's provided activity
<#
        }
#>
                <#= GetStateNames( transitionEnumerator.Current.TransitionTo ).EntryMethod #>();
<#
    }

    if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>

                if ( <#= GetUniqueName( "States" ) #>.Transition < <#= GetUniqueName( "currentState" )
#> )
                {
                    <#= GetUniqueName( "stateMachineThread" ) #> = new Thread( <#= GetUniqueName(
"StateMachineMethod" ) #> );
                    <#= GetUniqueName( "stateMachineThread" ) #>.Start();
                }
            }
<#
    }
    else
    {
#>
            }
            }
```

```
            finally
            {
                lock ( <#= GetUniqueName( "eventQueue" ) #> )
                {
                    if ( Thread.CurrentThread != <#= GetUniqueName( "eventQueue" ) #>.Dequeue() )
                        throw new InvalidOperationException( "Abnormal event processing! This exception
must never been thrown!" );
                    Monitor.PulseAll( <#= GetUniqueName( "eventQueue" ) #> );
                }
            }
<#
    }
#>
        }

<#
    if ( "" != this.StateMachine.CodeGeneration.JoinMethod )
    {
#>
        <#= this.StateMachine.CodeGeneration.JoinMethodModifiers #> void <#=
this.StateMachine.CodeGeneration.JoinMethod #>()
        {
<#
        if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
        {
#>
            Thread thread = null;

            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
                thread = <#= GetUniqueName( "stateMachineThread" ) #>;

            if ( null != thread )
                thread.Join();
<#
        }
        else
        {
#>
            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
            {
                while ( <#= GetUniqueName( "States" ) #>.Final != <#= GetUniqueName( "currentState" )
#> )
                    Monitor.Wait( <#= GetUniqueName( "stateMachineLockObject" ) #> );
            }
<#
        }
#>
        }
<#
    }
#>

<#
    if ( "" != this.StateMachine.CodeGeneration.AbortMethod )
    {
#>
        <#= this.StateMachine.CodeGeneration.AbortMethodModifiers #> void <#=
this.StateMachine.CodeGeneration.AbortMethod #>()
        {
            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
            {
                if ( <#= GetUniqueName( "States" ) #>.Final == <#= GetUniqueName( "currentState" ) #> )
                    return;
<#
        if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
        {
#>

                if ( null != <#= GetUniqueName( "stateMachineThread" ) #> )
                    <#= GetUniqueName( "stateMachineThread" ) #>.Abort();
<#
        }
#>

                <#= GetFinalStateNames().EntryMethod #>();
<#
        if ( Synchronization.Sync_NoThread == this.StateMachine.CodeGeneration.EventHandling )
        {
#>

                Monitor.PulseAll( <#= GetUniqueName( "stateMachineLockObject" ) #> );
```

```
<#
        }
#>
            }
        }
<#
    }

    foreach ( BaseState baseState in this.StateMachine.States )
    {
        State state = baseState as State;
        if ( null != state )
        {
#>

        private void <#= GetStateNames( baseState ).EntryMethod #>()
        {
            System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] entering state <#= baseState.Name #>...",
"StateMachineTrace" );
            <#= GetUniqueName( "currentState" ) #> = <#= GetUniqueName( "States" ) #>.<#=
GetStateNames( baseState ).StateName #>;
<#
            if ( "" != state.EntryActivity )
            {
#>
            System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing entry activity '<#= state.EntryActivity
#>'...", "StateMachineTrace" );
            <#= state.EntryActivity #> // User's provided activity
<#
            }
#>
        }

        private void <#= GetStateNames( baseState ).ExitMethod #>()
        {
<#
            if ( "" != state.ExitActivity )
            {
#>
            System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] producing exit activity '<#= state.ExitActivity
#>'...", "StateMachineTrace" );
            <#= state.ExitActivity #> // User's provided activity
<#
            }
#>
            <#= GetUniqueName( "currentState" ) #> = <#= GetUniqueName( "States" ) #>.Transition;
            System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] exiting state <#= baseState.Name #>...",
"StateMachineTrace" );
        }
<#
        }
    }
#>

        private void <#= GetFinalStateNames().EntryMethod #>()
        {
            <#= GetUniqueName( "currentState" ) #> = <#= GetUniqueName( "States" ) #>.Final;
            <#= GetUniqueName( "UnsubscribeEvents" ) #>();

<#
        if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
        {
#>
            <#= GetUniqueName( "stateMachineThread" ) #> = null;
<#
        }
#>

            System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] StateMachine is stopping...", "StateMachineTrace"
);
        }
<#
    foreach ( Event oEvent in this.StateMachine.CodeGeneration.Events )
    {
        if ( 0 < oEvent.Transitions.Count )
        {
```

```
#>

        private <#= oEvent.Return #> <#= GetEventNames( oEvent ).EventHandlerMethod #>( <#=
oEvent.Parameters #> )
        {
            System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class #> [ " +
Thread.CurrentThread.ManagedThreadId.ToString() + "] processing event <#= oEvent.Access #>...",
"StateMachineTrace" );

            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
            {
                switch ( <#= GetUniqueName( "currentState" ) #> )
                {
<#
                Dictionary<BaseState, List<Transition>> transitionsByState = GroupTransitionsByState(
oEvent );
                foreach ( BaseState baseState in transitionsByState.Keys )
                {
#>
                    case <#= GetUniqueName( "States" ) #>.<#= GetStateNames( baseState ).StateName #>:
<#
                    transitionEnumerator = transitionsByState[ baseState ].GetEnumerator();
                    transitionEnumerator.MoveNext();
                    if ( "" != transitionEnumerator.Current.Guard )
                    {
#>
                        if ( <#= transitionEnumerator.Current.Guard #> )
                        {
                            System.Diagnostics.Trace.WriteLine( "<#=
this.StateMachine.CodeGeneration.Class #> [ " + Thread.CurrentThread.ManagedThreadId.ToString() + "]
guard '<#= transitionEnumerator.Current.Guard #>' is true...", "StateMachineTrace" );

                            <#= GetStateNames( transitionEnumerator.Current.TransitionFrom ).ExitMethod
#>();
<#
                        if ( "" != transitionEnumerator.Current.Activity )
                        {
#>
                            System.Diagnostics.Trace.WriteLine( "<#=
this.StateMachine.CodeGeneration.Class #> [ " + Thread.CurrentThread.ManagedThreadId.ToString() + "]
producing transition activity '<#= transitionEnumerator.Current.Activity #>'...", "StateMachineTrace"
);
                            <#= transitionEnumerator.Current.Activity #> // User's provided activity
<#
                        }
#>
                            <#= GetStateNames( transitionEnumerator.Current.TransitionTo ).EntryMethod
#>();
                        }
<#
                        while ( transitionEnumerator.MoveNext() )
                        {
#>
                        else if ( <#= transitionEnumerator.Current.Guard #> )
                        {
                            System.Diagnostics.Trace.WriteLine( "<#=
this.StateMachine.CodeGeneration.Class #> [ " + Thread.CurrentThread.ManagedThreadId.ToString() + "]
guard '<#= transitionEnumerator.Current.Guard #>' is true...", "StateMachineTrace" );

                            <#= GetStateNames( transitionEnumerator.Current.TransitionFrom ).ExitMethod
#>();
<#
                            if ( "" != transitionEnumerator.Current.Activity )
                            {
#>
                            System.Diagnostics.Trace.WriteLine( "<#=
this.StateMachine.CodeGeneration.Class #> [ " + Thread.CurrentThread.ManagedThreadId.ToString() + "]
producing transition activity '<#= transitionEnumerator.Current.Activity #>'...", "StateMachineTrace"
);
                            <#= transitionEnumerator.Current.Activity #> // User's provided activity
<#
                            }
#>
                            <#= GetStateNames( transitionEnumerator.Current.TransitionTo ).EntryMethod
#>();
                        }
<#
                        }
                    }
                    else
                    {
```

```
#>
                        <#= GetStateNames( transitionEnumerator.Current.TransitionFrom ).ExitMethod
#>();
<#
                    if ( "" != transitionEnumerator.Current.Activity )
                    {
#>
                        System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class
#> [" + Thread.CurrentThread.ManagedThreadId.ToString() + "] producing transition activity '<#=
transitionEnumerator.Current.Activity #>'...", "StateMachineTrace" );
                        <#= transitionEnumerator.Current.Activity #> // User's provided activity
<#
                    }
#>
                        <#= GetStateNames( transitionEnumerator.Current.TransitionTo ).EntryMethod
#>();
<#
                }
#>
                        break;
<#
            }
#>
                    default:
                        System.Diagnostics.Trace.WriteLine( "<#= this.StateMachine.CodeGeneration.Class
#> [" + Thread.CurrentThread.ManagedThreadId.ToString() + "] StateMachine <#= oEvent.Access #> has not
being processed...", "StateMachineTrace" );
                        break;
                }
<#
            if ( Synchronization.Sync_NoThread == this.StateMachine.CodeGeneration.EventHandling )
            {
#>

                Monitor.PulseAll( <#= GetUniqueName( "stateMachineLockObject" ) #> );
<#
            }
#>
            }
        }
<#
        }
    }
#>

<#
    if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>
        private enum <#= GetUniqueName( "Events" ) #>
        {
<#
        foreach ( Event oEvent in this.StateMachine.CodeGeneration.Events )
        {
            if ( 0 < oEvent.Transitions.Count )
            {
#>
            <#= GetEventNames( oEvent ).EventName #>,
<#
            }
        }
#>
        }

        private Queue< KeyValuePair< <#= GetUniqueName( "Events" ) #>, object[] > > <#= GetUniqueName(
"eventQueue" ) #> = new Queue< KeyValuePair< <#= GetUniqueName( "Events" ) #>, object[] > >();
<#
    }
#>
<#
    foreach ( Event oEvent in this.StateMachine.CodeGeneration.Events )
    {
        if ( 0 < oEvent.Transitions.Count )
        {
#>

        private <#= oEvent.Return #> <#= GetEventNames( oEvent ).EventSubscribeMethod #>( <#=
oEvent.Parameters #> )
        {
<#
```

```
            List< KeyValuePair< string, string > > parsedParameters = ParseParameters(
oEvent.Parameters );
            IEnumerator< KeyValuePair< string, string > > parametersEnumerator =
parsedParameters.GetEnumerator();

            if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
            {
#>
            lock ( <#= GetUniqueName( "eventQueue" ) #> )
            {
                <#= GetUniqueName( "eventQueue" ) #>.Enqueue( new KeyValuePair< <#= GetUniqueName(
"Events" ) #>, object[] >(
                    <#= GetUniqueName( "Events" ) #>.<#= GetEventNames( oEvent ).EventName #>,
                    new object[] { <#
                if ( parametersEnumerator.MoveNext() )
                {
                    #><#= parametersEnumerator.Current.Value #><#
                    while ( parametersEnumerator.MoveNext() )
                    {
                        #>, <#= parametersEnumerator.Current.Value #><#
                    }
                }
#> } ) );

                Monitor.PulseAll( <#= GetUniqueName( "eventQueue" ) #> );
            }
<#
            }
            else
            {
#>
            lock ( <#= GetUniqueName( "eventQueue" ) #> )
            {
                if ( ( 0 < <#= GetUniqueName( "eventQueue" ) #>.Count ) &&
                     ( Thread.CurrentThread == <#= GetUniqueName( "eventQueue" ) #>.Peek() ) )
                    throw new InvalidOperationException( "Processed event can't produce events!" );

                <#= GetUniqueName( "eventQueue" ) #>.Enqueue( Thread.CurrentThread );
                while ( Thread.CurrentThread != <#= GetUniqueName( "eventQueue" ) #>.Peek() )
                    Monitor.Wait( <#= GetUniqueName( "eventQueue" ) #> );
            }

            try
            {
                <#= GetEventNames( oEvent ).EventHandlerMethod #>( <#
                if ( parametersEnumerator.MoveNext() )
                {
                    #><#= parametersEnumerator.Current.Value #><#
                    while ( parametersEnumerator.MoveNext() )
                    {
                        #>, <#= parametersEnumerator.Current.Value #><#
                    }
                }
#> );
            }
            finally
            {
                lock ( <#= GetUniqueName( "eventQueue" ) #> )
                {
                    if ( Thread.CurrentThread != <#= GetUniqueName( "eventQueue" ) #>.Dequeue() )
                        throw new InvalidOperationException( "Abnormal event processing! This exception
must never been thrown!" );
                    Monitor.PulseAll( <#= GetUniqueName( "eventQueue" ) #> );
                }
            }
<#
            }
#>
        }
<#
        }
    }

    if ( Synchronization.Async_OwnThread == this.StateMachine.CodeGeneration.EventHandling )
    {
#>

        private void <#= GetUniqueName( "StateMachineMethod" ) #>()
        {
            <#= GetUniqueName( "States" ) #> currentState;
```

```
            lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
                currentState = <#= GetUniqueName( "currentState" ) #>;

            while ( <#= GetUniqueName( "States" ) #>.Final != currentState )
            {
                KeyValuePair< <#= GetUniqueName( "Events" ) #>, object[] > eventAndParameters;

                lock ( <#= GetUniqueName( "eventQueue" ) #> )
                {
                    while ( 0 == <#= GetUniqueName( "eventQueue" ) #>.Count )
                        Monitor.Wait( <#= GetUniqueName( "eventQueue" ) #> );
                    eventAndParameters = <#= GetUniqueName( "eventQueue" ) #>.Dequeue();
                }

                switch ( eventAndParameters.Key )
                {
<#
        foreach ( Event oEvent in this.StateMachine.CodeGeneration.Events )
        {
            if ( 0 < oEvent.Transitions.Count )
            {
                List< KeyValuePair< string, string > > parsedParameters = ParseParameters(
oEvent.Parameters );
                IEnumerator< KeyValuePair< string, string > > parametersEnumerator =
parsedParameters.GetEnumerator();
#>
                    case <#= GetUniqueName( "Events" ) #>.<#= GetEventNames( oEvent ).EventName #>:
                        <#= GetEventNames( oEvent ).EventHandlerMethod #>( <#
                if ( 0 < parsedParameters.Count )
                {
                    #>(<#= parsedParameters[ 0 ].Key #>) eventAndParameters.Value[ 0 ]<#
                    for ( int i = 1; i < parsedParameters.Count; i++ )
                    {
                        #>, (<#= parsedParameters[ i ].Key #>) eventAndParameters.Value[ <#=
i.ToString() #> ]<#
                    }
                }
#> );
                        break;
<#
            }
        }
#>
                }

                lock ( <#= GetUniqueName( "stateMachineLockObject" ) #> )
                    currentState = <#= GetUniqueName( "currentState" ) #>;
            }
        }
<#
    }
#>

        private void <#= GetUniqueName( "SubscribeEvents" ) #>()
        {
<#
    foreach ( Event oEvent in this.StateMachine.CodeGeneration.Events )
    {
        if ( 0 < oEvent.Transitions.Count )
        {
#>
            <#= oEvent.Access #> += <#= GetEventNames( oEvent ).EventSubscribeMethod #>;
<#
        }
    }
#>
        }

        private void <#= GetUniqueName( "UnsubscribeEvents" ) #>()
        {
<#
    foreach ( Event oEvent in this.StateMachine.CodeGeneration.Events )
    {
        if ( 0 < oEvent.Transitions.Count )
        {
#>
            <#= oEvent.Access #> -= <#= GetEventNames( oEvent ).EventSubscribeMethod #>;
<#
        }
    }
#>
```

```
            }
        }
    }
```