САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ МЕХАНИКИ И ОПТИКИ

Кафедра «Компьютерных технологий»

Е. О. Решетников

Инструментальное средство для визуального проектирования автоматных программ на основе *Microsoft Domain-Specific Language Tools*

Бакалаврская работа

Руководитель: докт. техн. наук, профессор Шалыто А. А.

Санкт-Петербург

2007

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время в связи с большим числом используемых и разрабатываемых стандартов и технологий программирования стало ясно, что попытка создать единый универсальный стандарт построения и взаимодействия программных систем обречена на неудачу. Концепция *Model Driven Architecture (MDA)* [1] призвана обеспечить общее основание для описания и применения большинства существующих стандартов.

Использование *MDA* не ограничивает разработчиков в выборе конкретных технологий. Интеграция стандартов достигается за счет введения платформо-независимой модели приложения, а также унифицированного языка для описания таких моделей. В качестве последнего выступает *Unified Modeling Language (UML)* [2]. Диаграммы языка *UML* позволяют описывать как модель целого приложения, так и модель внутренней организации приложения, а также принципы взаимодействия отдельных его частей.

Нельзя не отметить тенденцию роста числа продуктов, которые помогают автоматизировать разработку программного обеспечения. При этом по модели приложения часть кода генерируется автоматически. Построение моделей, которые наиболее хорошо отображают поведение системы, может значительно увеличить часть автоматически генерируемого кода.

Среда разработки *Microsoft Visual Studio* [3] является основной средой разработки программных продуктов для семейства операционных систем *Microsoft Windows*. Версия 2005 данной среды содержит несколько компонентов для построения моделей: дизайнер распределенных систем, дизайнер классов, а также дополнительные модули, позволяющие производить разработку собственных визуальных редакторов моделей.

Одними из наиболее часто используемых *UML*-диаграмм являются:

1. Диаграмма классов – статическая структурная диаграмма, описывающая зависимости между классами системы.

2. Диаграмма объектов – отображение экземпляров классов системы с указанием текущих значений их атрибутов и связей между ними.
3. Диаграмма автомата – модель для представления динамики исполнения приложения.

В данной работе описывается пример добавления в *Microsoft Visual Studio 2005* инструментального средства , позволяющего производить проектирование приложений с использованием трех вышеперечисленных *UML*-диаграмм. Это средство должно существенно упростить процесс разработки преимущественно «реактивных» приложений, а также уменьшить долю написанного вручную кода и, тем самым, уменьшить вероятность возникновения ошибок.

1. ПОСТАНОВКА ЗАДАЧИ

В 2006 году Алексей Ларионов, магистрант СПбГУ ИТМО кафедры компьютерных технологий, разработал инструментальное средство автоматного программирования [4]. Это средство интегрируется в среду разработки *Microsoft Visual Studio 2005* и расширяет ее возможности по проектированию и реализации программных продуктов.

Цель настоящей работы – расширить возможности разработанного в [4] инструментального средства.

Новое инструментальное средство также будет интегрироваться в среду разработки *Microsoft Visual Studio 2005*. Сама же стадия создания программных продуктов в разрабатываемом инструментальном средстве будет состоять из трех этапов:

1. Визуальное построение модели проекта: создание классов, интерфейсов, связей между ними. Для этого этапа будет использована разрабатываемая в данной работе диаграмма классов (причина, по которой не используется стандартная диаграмма классов среды *Visual Studio 2005*, обсуждена в разд.3.1).
2. Добавление автоматного поведения к некоторым классам: к любому выделенному классу на *Диаграмме Классов* может быть добавлено автоматное поведение. При этом будет использована *Диаграмма Автомата*.
3. Визуальное построение экземпляров объектов проекта: представление объектов на *Диаграмме Объектов*.

Построенные с помощью разрабатываемых диаграмм модели будут проверяться на валидность на основе встроенных средств проверки правильности моделей. В случае отсутствия ошибок валидации, по диаграммам будет генерироваться исходный код приложения на языке *C#* [5].

Разрабатываемое инструментальное средство должно позволять не только полностью разрабатывать программные продукты на его основе, но и использовать его на любой стадии разработки проекта.

2. ОБЗОР ПРОГРАММНЫХ ПРОДУКТОВ-АНАЛОГОВ

В данной главе рассмотрены некоторые программные продукты, позволяющие визуально проектировать модели программ с последующей автоматической генерацией кода по полученным моделям. В работе [4] описана часть таких продуктов. Поэтому рассмотрим продукты, которые либо не были рассмотрены в работе [4], либо были рассмотрены недостаточно.

2.1. Rational Rose

Продукт *Rational Rose* [6], выпущенный компанией *IBM Rational Software*, поддерживает визуальное объектно-ориентированное моделирование. При моделировании используется *UML*-нотация. Продукт поддерживает генерацию кода, а также обратное проектирование (построение модели по программному коду) для многих языков программирования. Позволяет строить объектную модель разрабатываемой системы, определять спецификации классов, объектов, атрибутов и операций.

Этот продукт поддерживает большинство основных *UML*-диаграмм для детального моделирования проекта. Несмотря на это, большинство поддерживаемых диаграмм никак не влияет на генерируемый код.

Существенным недостатком этого продукта является то, что он предназначен для генерации кода лишь декларативной части проекта. Генерируемый код отражает только иерархию классов спроектированного проекта, а также артефакты (поля, методы и т.д.) конкретных классов. Вся функциональность должна быть написана вручную разработчиком с учетом особенностей выбранного языка программирования, используемых технологий и т.д.

Внешний вид среды разработки *Rational Rose* для *Microsoft Visual Studio*, а также вид проектируемых моделей, представлены на рис. 1.

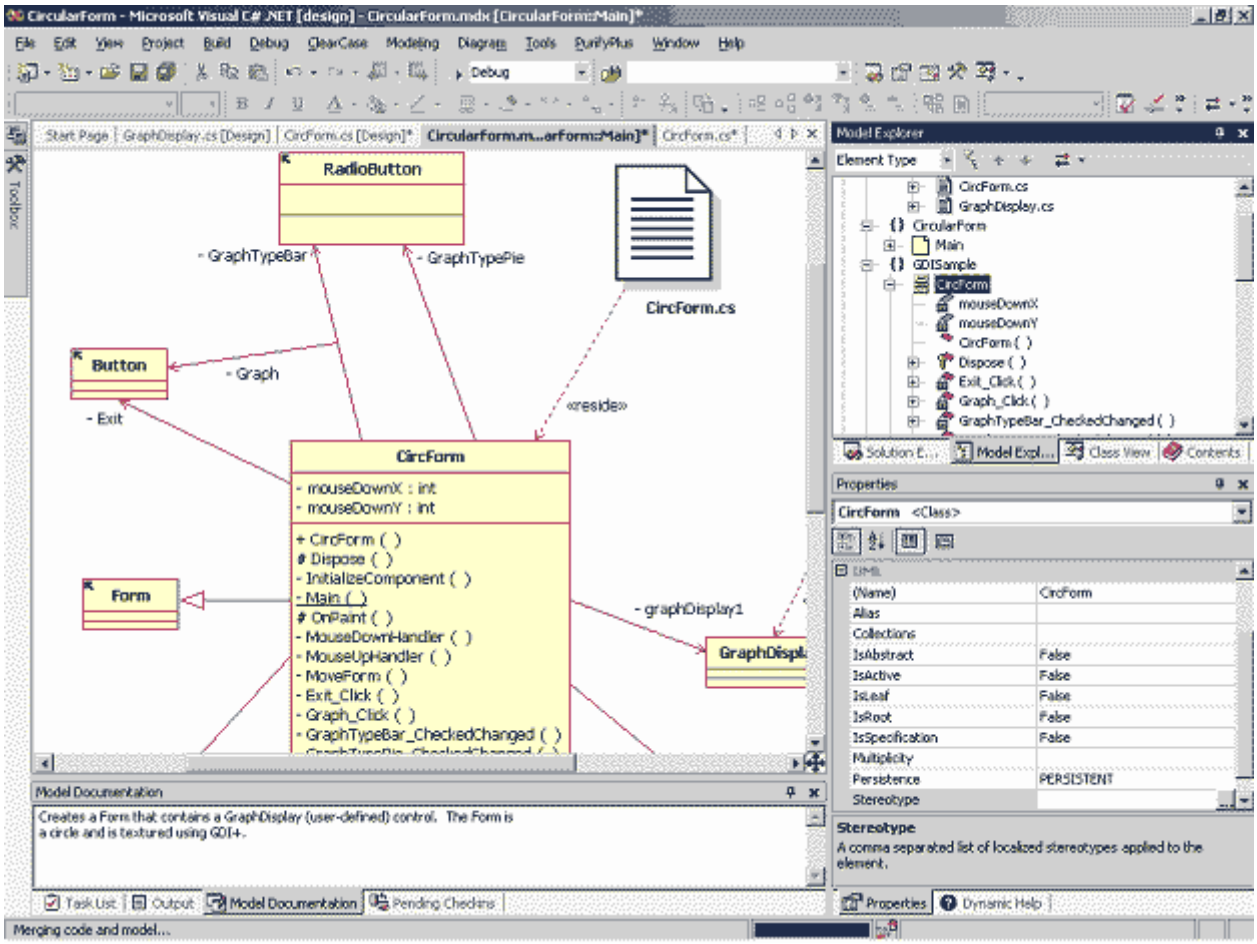

Рис. 1. Использование *Rational Rose* для *Visual Studio*

2.2. PowerDesigner

Этот продукт от компании *Sybase* [7] представляет собой полнофункциональный инструментарий для создания бизнес-приложений. Он включает в себя средства моделирования бизнес-процессов, возможности проектирования баз данных, моделирования приложения, а также генерации кода по полученной модели на языках *Java*, *C#*, *C++*, *PowerBuilder*, *VB.Net*.

Так как данный продукт направлен на разработку бизнес-приложений, использование его в качестве средства разработки произвольных программных продуктов не удобно. Генерация кода также сводится только к генерации декларативной части кода по диаграмме классов.

Рассмотренные продукты позволяют моделировать проект и генерировать часть кода, но не предоставляют никаких возможностей по разработке «реактивных» систем с использованием моделей конечных автоматов. Рассмотрим несколько продуктов, которые поддерживают разработку таких систем.

2.3. Enterprise Architect

Продукт компании *Sparx Systems* [8]. Сочетает в себе богатство нотаций последних версий *UML* с интуитивно понятным интерфейсом. Благодаря поддержке широкого спектра *UML*-диаграмм, с помощью данного продукта возможно описание практически любого приложения. Благодаря модели *Domain Model*, возможно детальное описание поведения приложения с помощью конечных автоматов.

Несмотря на то, что продукт продолжает обновляться и выходят все более новые версии, генерация кода по автоматной модели в нем отсутствует. Впрочем, генерации кода нет и по моделям других типов диаграмм. Сгенерировать код можно лишь для классов, созданных с помощью поддерживаемой диаграммы классов. Таким образом, хотя продукт и обладает богатыми возможностями по созданию моделей приложения, он не предназначен для разработки программ.

Также существенным недостатком продукта является его обособленность от сред разработки программного обеспечения. Даже в случае использования сгенерированного продуктом кода, необходимо его портирование в какую-либо среду разработки для дальнейшей реализации логики создаваемого приложения.

Внешний вид среды *Enterprise Architect* отображен на рис. 2.

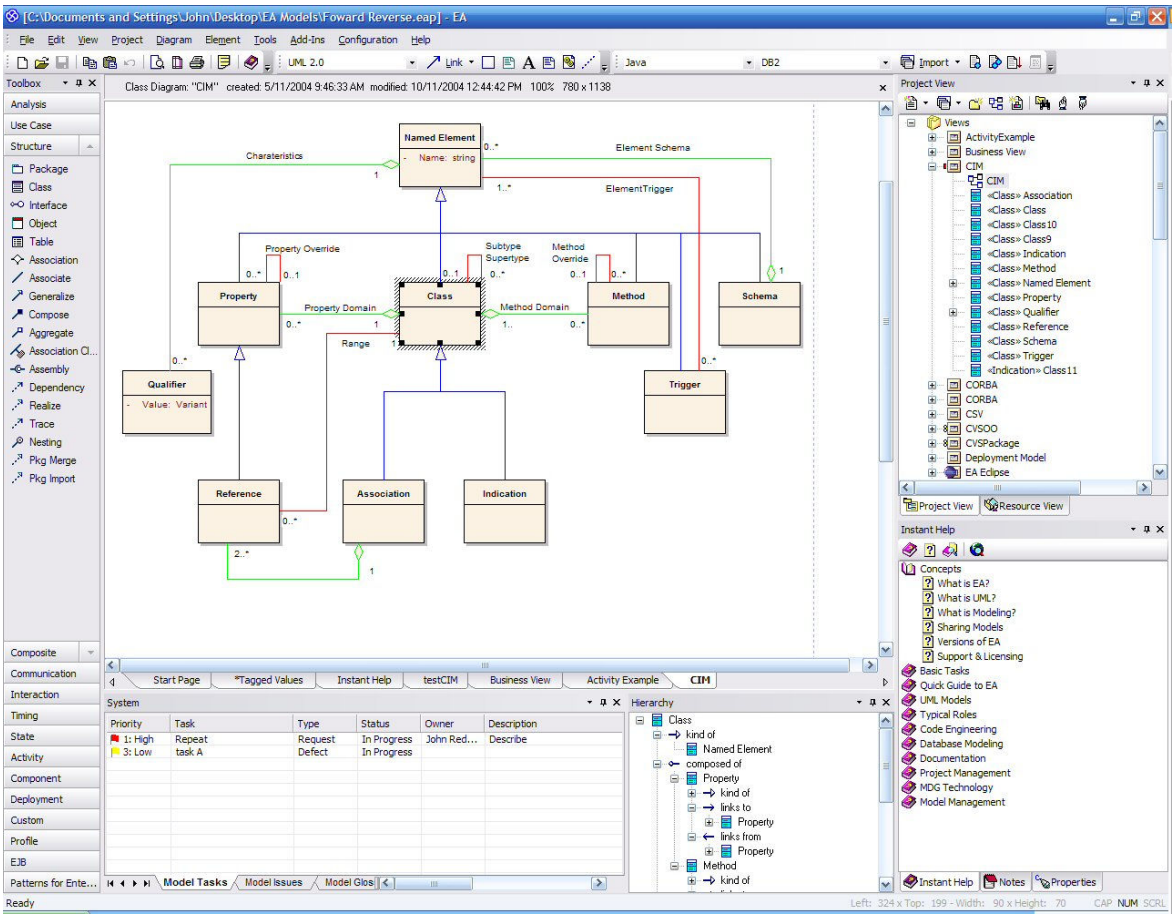

Рис. 2. Рабочая область инструмента моделирования *Enterprise Architect*

2.4. SmartState

Продукт от компании *ApeSoft* [9]. Этот продукт разработан специально для моделирования программ с автоматным поведением и генерации по полученной модели кода. С помощью данного продукта возможна генерация кода на языках *C++*, *Java*, *C*, *C#*.

Впрочем, как и все указанные выше продукты, *SmartState* представляет собой самостоятельное приложение. Таким образом, скомпилировать и запустить программу, полученную по сгенерированному коду, не представляется возможным. Полученный код должен быть портирован в одну из сред разработки программного обеспечения и последующая реализация приложения будет не связана со *SmartState*. Это является большим недостатком, если во время реализации, вдруг, окажется, что автоматная модель приложения должна быть незначительно изменена. В этом случае генерация кода должна быть запущена

еще раз, и вся логика, написанная вручную, должна быть заново внесена в сгенерированный код.

На рис. 3 изображен *SmartState* с визуальной моделью автомата.

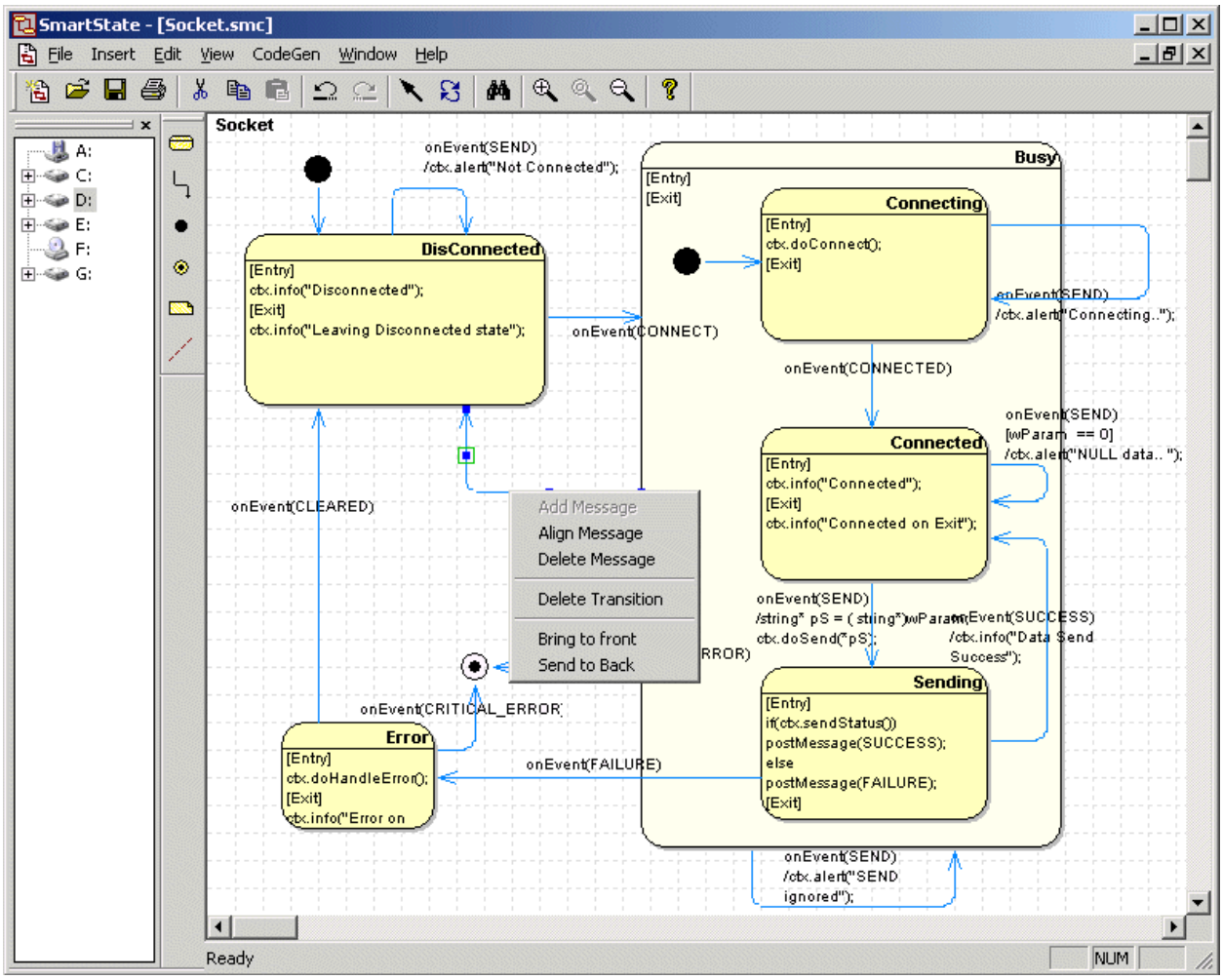

Рис. 3. Разработка автоматной модели в *SmartState*

2.5. Windows Workflow Foundation

Продукт *Windows Workflow Foundation* [10] от корпорации *Microsoft* является на данный момент самым мощным продуктом, позволяющим производить визуальное проектирование моделей приложений и генерацию кода по полученным моделям. Несмотря на то, что *Windows Workflow Foundation* должен облегчить разработку приложений бизнес-процессов, он содержит также богатый инструментарий для создания приложений самого различного типа. *State Machine Workflow* – один из компонентов для визуального проектирования

приложений с использованием диаграмм состояний. На рис. 4 изображен пример создания автоматной модели в *State Machine Workflow*.

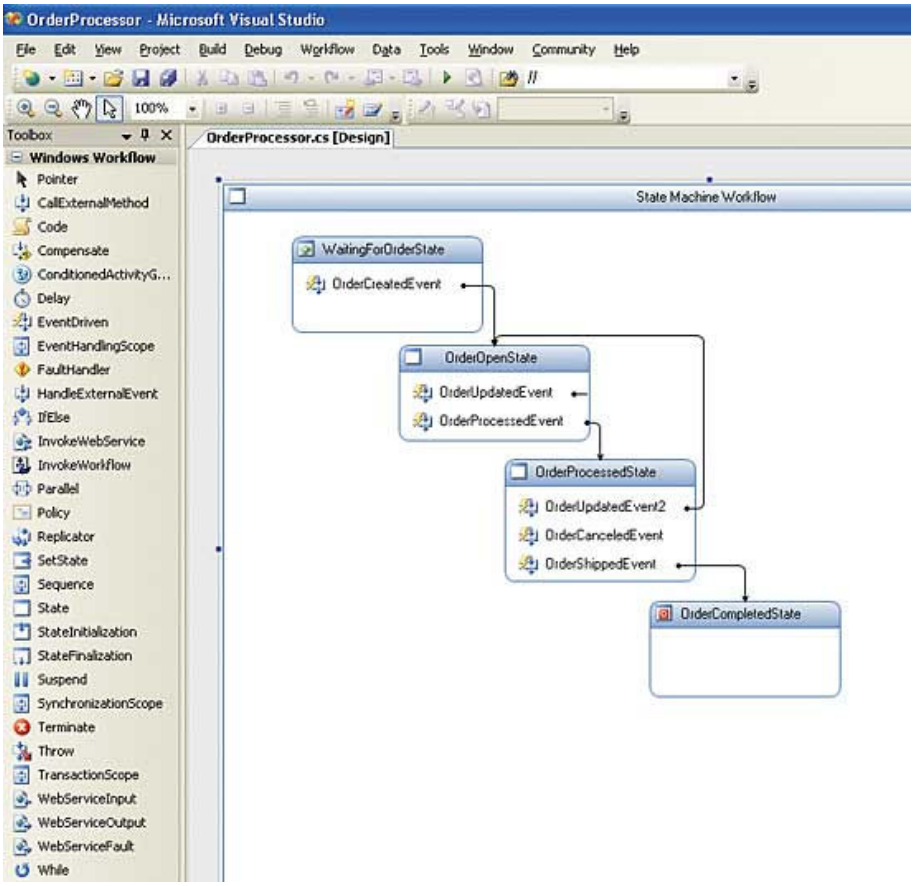

Рис. 4. Создание автоматной модели в *State Machine Workflow*

Переходы между состояниями автомата также осуществляются при возникновении некоторого события. По полученной модели возможна генерация кода на любом языке семейства *.Net*. Автоматная модель еще одного приложения, разработанного в *State Machine Workflow*, показана на рис. 5.

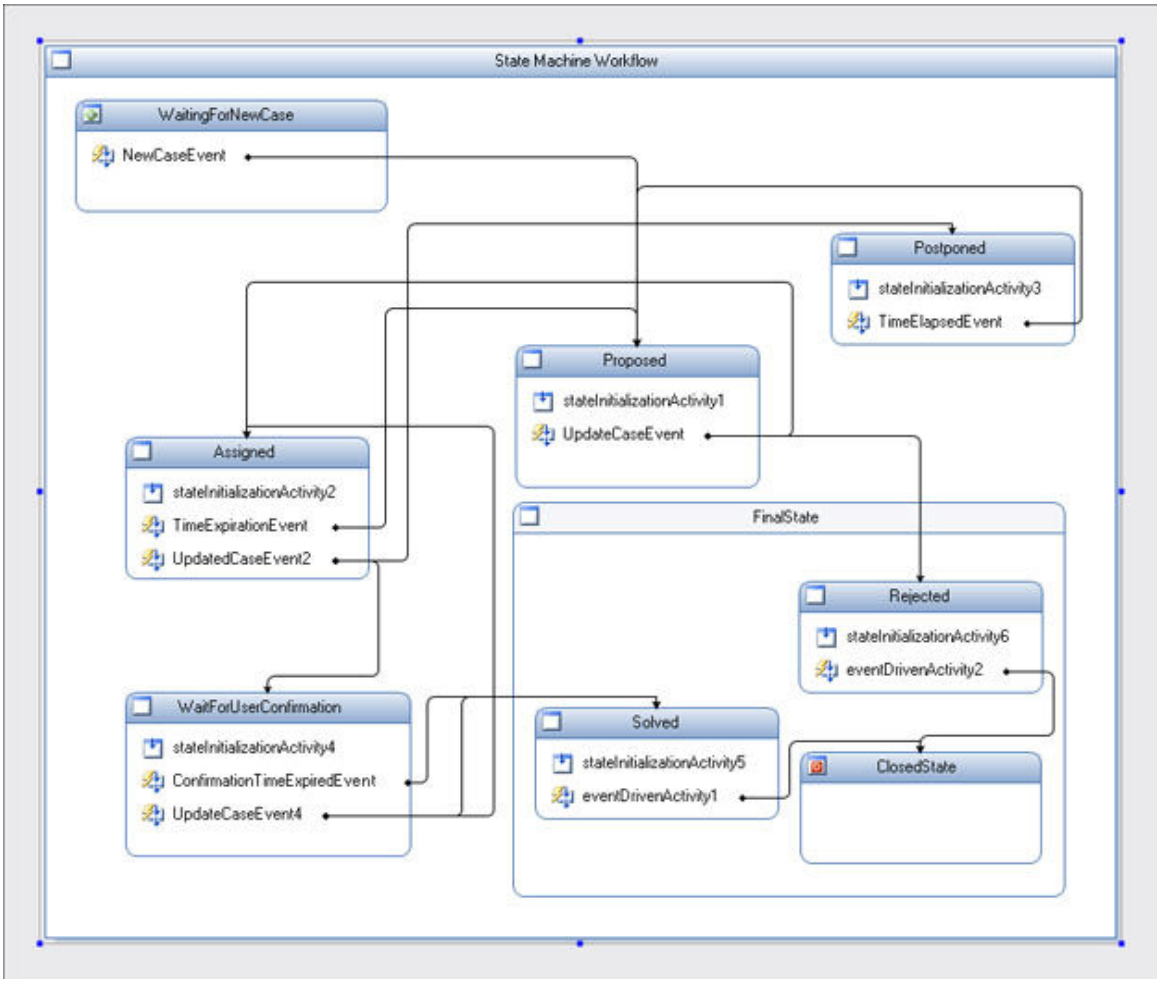

Рис. 5. Модель автомата в *State Machine Workflow*

Продукт *Windows Workflow Foundation* не имеет таких явных недостатков, как предыдущие продукты, однако стоит отметить некоторые аспекты, которые могут помешать быстрой и простой разработке программных продуктов с помощью *Windows Workflow Foundation*. Так как продукт имеет очень большие возможности и направлен на решение широкого круга задач, он имеет впечатляющую коллекцию инструментов для визуального проектирования моделей программ. Поэтому у разработчика уйдет немало времени для того, чтобы разобраться в том, как правильно проектировать программы, используя данный продукт. Интерфейс программы является довольно привычным для разработчиков программного обеспечения в средах *Microsoft Visual Studio*, но обилие инструментов для разработки и схожесть в их названиях, не могут первое время не препятствовать разработке приложений.

Продукт *Windows Workflow Foundation* пока не входит в стандартную поставку *Microsoft Visual Studio 2005* и может быть использован в данной среде разработки только при установки дополнительных компонентов в качестве расширения среды (*Visual Studio 2005 Extensions for Windows Workflow Foundation*). В поставку же он войдет только в новой версии *Microsoft Visual Studio "Orcas"* [11].

2.6. State Machine Designer

Этот продукт представлен в работе [4] и является основой для разрабатываемого в настоящей работе инструментального средства. Продукт *State Machine Designer* позволяет проектирование моделей автоматных систем при помощи диаграмм состояний. Недостатком продукта является отсутствие возможности моделирования статической модели приложения наряду с моделированием диаграммы состояний. В этом продукте отсутствует возможность визуального создания объектов приложения, как отсутствует и сама *Диаграмма Объектов*, которая является нововведением, реализованным в разрабатываемом инструментальном средстве.

Подводя итог обзора продуктов-аналогов, можно также упомянуть инструментальное средство *Unimod* [12]. Данный продукт обеспечивает визуальное построение объектно-ориентированных программ, поведение которых описывается диаграммами состояний. Генерация исполняемого кода по этим диаграммам выполняется на языке *Java*. Так как продукт является плагином к среде разработки программного обеспечения *Eclipse*, то при его использовании неудобств при реализации логики приложения не возникает. Генерация кода по модели происходит синхронно с ее редактированием. Разработчики *Unimod* в настоящее время работают над второй версией продукта. В новой версии должны появиться дополнительные возможности для визуальной разработки приложений, а также будет автоматизирован процесс верификации проектируемых моделей.

3. РЕАЛИЗАЦИЯ

Данная работа, как и работа [4], которая положена в ее основу, реализована при помощи *Инструментов Специализированных Языков Предметной Области* (*Domain-Specific Language Tools, DSL Tools*) [13], входящих в состав *Microsoft Visual Studio 2005 SDK*.

Специализированный язык предметной области (*Domain-Specific Language*) – это язык, разработанный для того, чтобы быть полезным для решения узкого круга специфических задач. С помощью инструментов *DSL Tools* возможно создание специализированных инструментов моделирования путем определения нового языка моделирования и его реализации.

Инструменты Специализированных Языков Предметной Области могут быть использованы для построения собственных визуальных редакторов, приспособленных для конкретной предметной области. Для этого необходимо создание метамодели языка создаваемого редактора при помощи *Специализированного Языка Предметной Области*. Используя полученную метамодель, в работе создается визуальный редактор. Также с использованием созданной метамодели возможна генерация какого-либо отчета по создаваемой в новом редакторе модели. Для моделей, описывающих конструкции языков программирования, возможна генерация исходного кода.

Остановимся на реализации метамоделей диаграммы объектов, диаграммы классов, а также диаграммы автомата.

3.1. Диаграмма классов

Диаграмма классов предназначена для визуального проектирования классов и интерфейсов разрабатываемого проекта. Среда разработки *Microsoft Visual Studio 2005* имеет собственную диаграмму классов, в которой также можно визуально создавать классы. Недостатком данной диаграммы является то, что она не всегда остается синхронизированной с исходным кодом. К примеру, если создать на диаграмме классов класс, а потом удалить диаграмму, созданный класс

все равно останется в проекте. И наоборот, если создать класс отдельно от диаграммы, то по умолчанию, новый класс на диаграмме не появится. Компания *Borland* в своем проекте *Together* [14] делает очень большой акцент на синхронизации визуальных моделей приложения и исходного кода. При этом разрабатываемая диаграмма классов должна полностью отражать код, который будет по ней сгенерирован. Рассмотренная причина, по которой не используется стандартная диаграмма классов, не является единственной. Также важным аспектом является наличие возможности добавления для некоторых классов автоматного поведения. Это означает, что если разрабатывается класс объекта, поведение которого будет описано с помощью конечного автомата, то на диаграмме классов это тоже должно быть отражено. Более того, должна быть реализована возможность перехода от диаграммы классов к диаграмме автомата для конкретного класса.

3.1.1. Метамодель диаграммы классов

На рис. 6 отображена метамодель диаграммы классов, разработанная при помощи *Инструментов Специализированных Языков Предметной Области* в среде разработки *Microsoft Visual Studio 2005*.

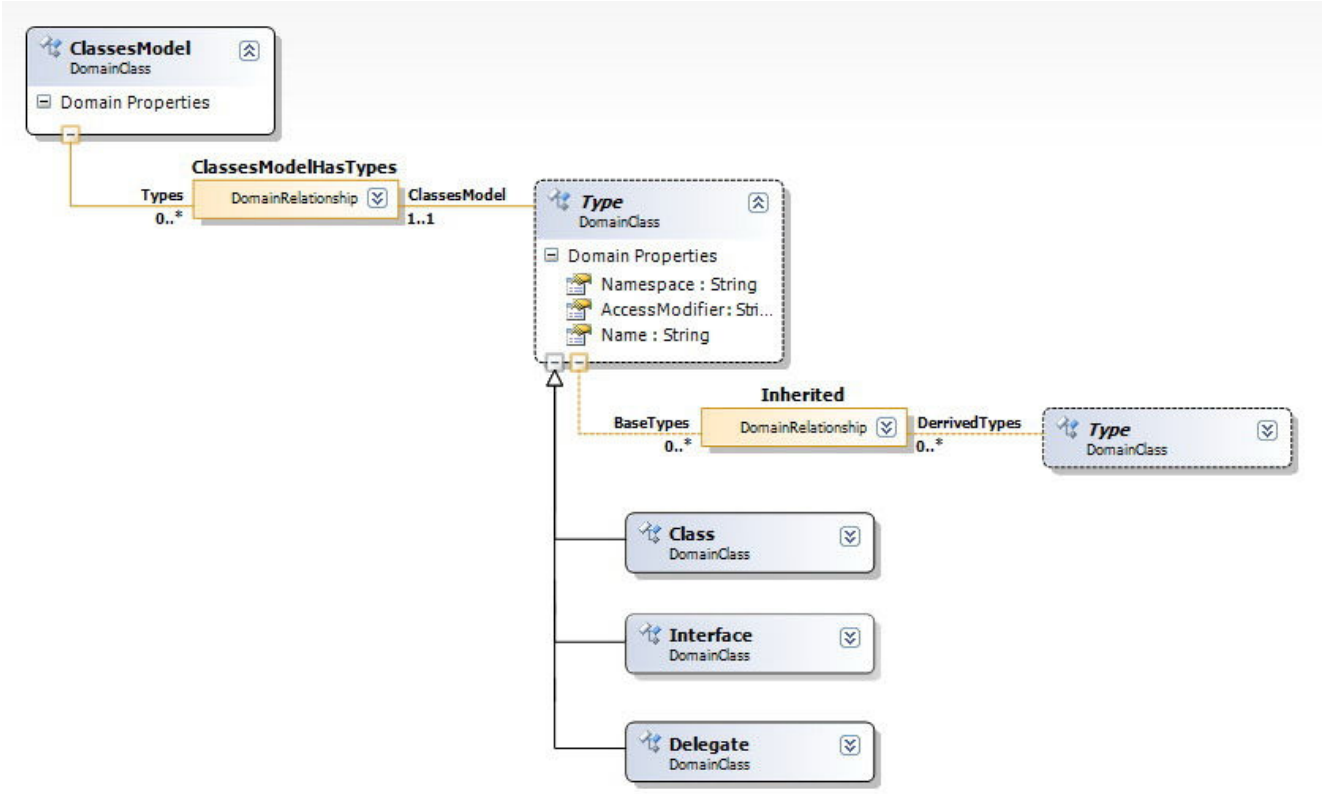

Рис. 6. Метамодель диаграммы классов

На метамодели диаграммы классов *ClassesModel* корневой элемент отображает саму диаграмму. Отношение *ClassesModelHasTypes* означает, что на диаграмме будут расположены объекты типа *Type*. Класс *Type* – базовый класс для классов *Class*, *Interface* и *Delegate*, которые отображают класс, интерфейс и делегат языка *C#* соответственно. Отношение *Inherited* означает наследование – объект типа *Type* может наследоваться от другого объекта типа *Type*. На рис. 7 представлена метамодель для класса.

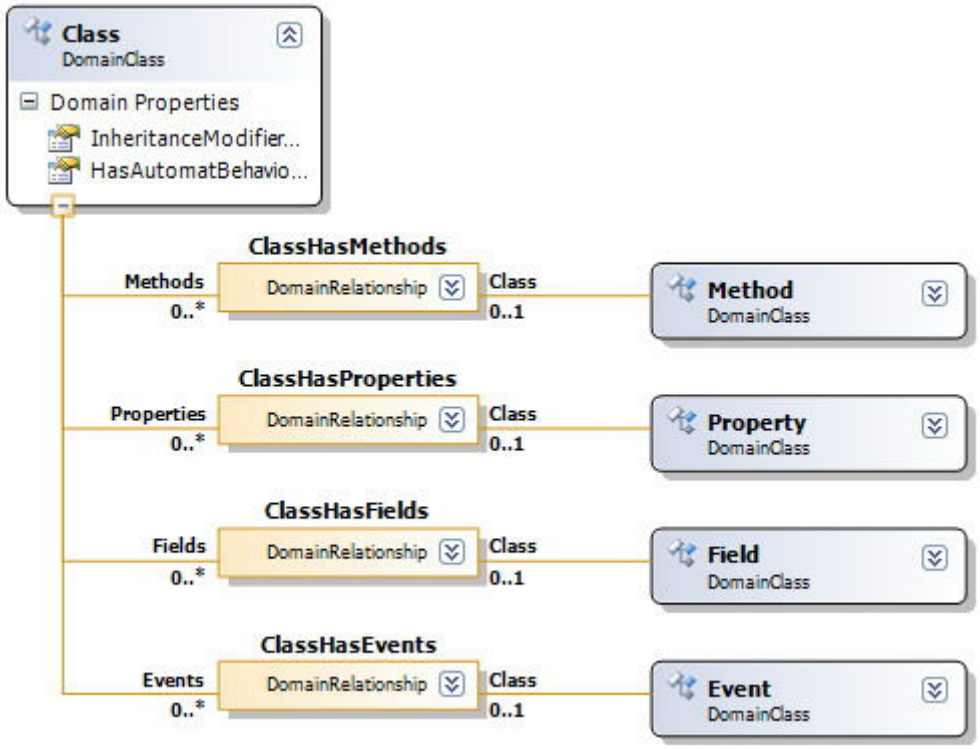

Рис. 7. Метамодель класса

Отношения *ClassHasMethods*, *ClassHasProperties*, *ClassHasFields*, *ClassHasEvents* означают, что класс может иметь *Методы*, *Свойства*, *Поля* и *События*.

На рис. 8 представлена метамодель интерфейса.

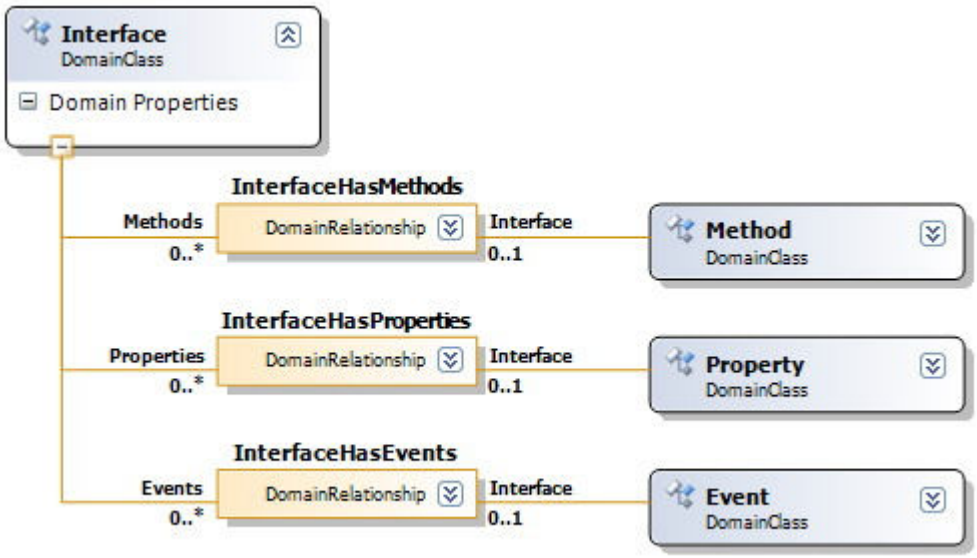

Рис. 8. Метамодель интерфейса

Отношения *InterfaceHasMethods*, *InterfaceHasProperties*, *InterfaceHasEvents* означают, что интерфейс может иметь *Методы*, *Свойства* и *События*. Заметим, что, в отличие от класса, у интерфейса не может быть полей.

Сами же *Методы*, *Свойства*, *События* и *Поля* являются членами класса или интерфейса и тоже имеют свои атрибуты. На рис. 9 отображена метамодель для членов класса.

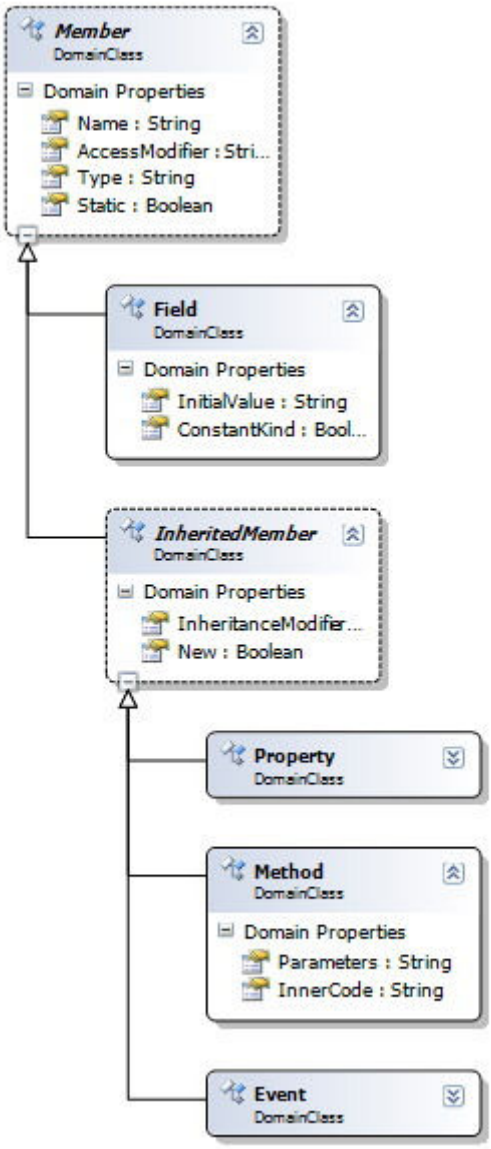

Рис. 9. Метамодель для членов класса и интерфейса

На метамодели членов класса отображены свойства, которыми они могут обладать. Например, все они имеют имя, модификатор доступа, тип. Для *Поля*

также можно задать начальное значение и модификатор константности. *Метод* имеет список параметров, а также исходный код метода.

На рис. 10 представлена метамодель для делегата.

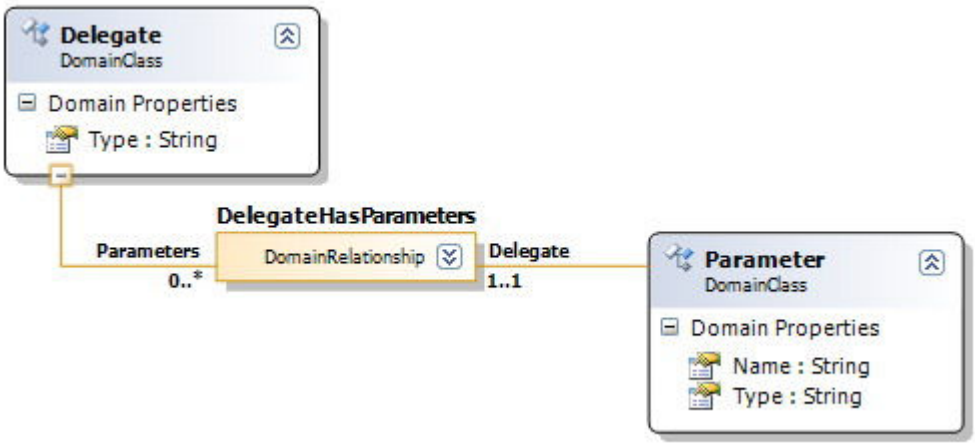

Рис. 10. Метамодель делегата

Отношение *DelegateHasParameters* означает, что у делегата может быть несколько параметров. Для каждого параметра задается имя параметра и его тип.

Таким образом, метамодель описывает модель диаграммы классов. Основной код *DSL*-проекта генерируется по файлам, определяющим язык. После изменения метамодели и запуска трансформации текстовых темплейтов, по метамодели будет сгенерирован код, с помощью которого будет возможно визуальное создание разработанной диаграммы классов.

3.1.2. Валидация диаграммы классов

При разработке метамодели учитываются основные возможности и особенности языка. Однако на ней невозможно отобразить всю семантику разрабатываемого языка. Например, наличие разработанной метамодели диаграммы классов, не запрещает наследовать интерфейс от класса, что невозможно в любом языке программирования. Поэтому для диаграммы классов дополнительно реализован ряд проверок на правильность проектируемой модели.

Ниже перечислены правила, которые учитываются при валидации диаграммы классов:

1. Интерфейс не может быть унаследован от класса.
2. Класс может быть унаследован не более чем от одного класса.
3. Класс не должен иметь модификатор доступа выше модификатора доступа базового класса.
4. В иерархии наследования не может быть циклов.
5. Все классы и интерфейсы должны иметь уникальные имена.

В случае невыполнения какого-либо правила в момент проектирования диаграммы классов, будет выдано соответствующее сообщение об ошибке.

3.2. Диаграмма автомата

Диаграмма автомата предназначена для визуального построения модели поведения объекта в случае, когда «жизненный цикл» объекта может быть представлен в виде выделенных состояний и переходов между ними, осуществляемых по «охраняемым» событиям [15].

3.2.1. Метамодель диаграммы автомата

Метамодель диаграммы автомата изображена на рис. 11.

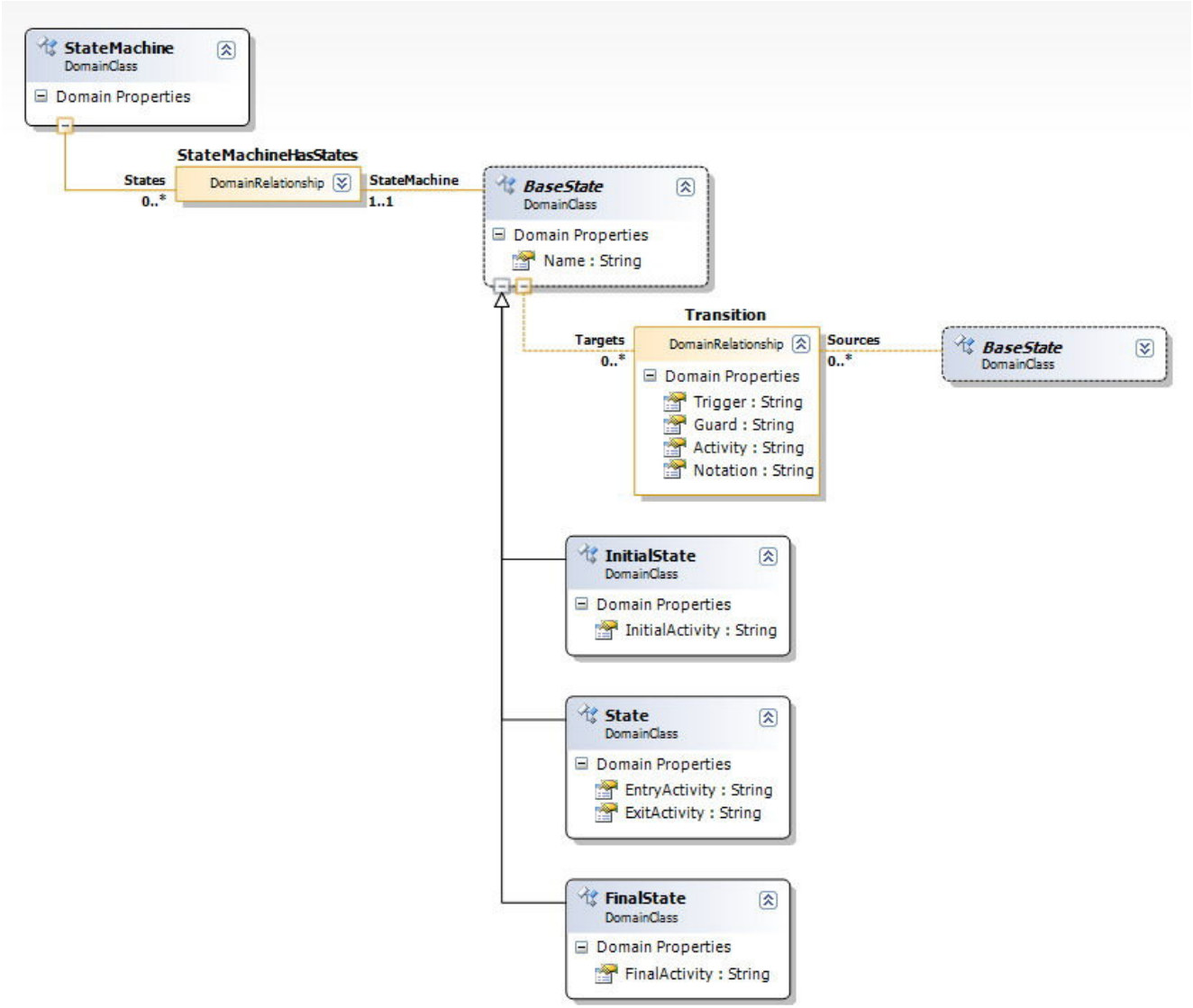

Рис. 11. Метамодель диаграммы автомата

Опишем метамодель диаграммы автомата. Корневым элементом метамодели является сама диаграмма автомата *StateMachine*. Отношение *StateMachineHasStates* означает, что автомат имеет состояния. *BaseState* – базовый тип для состояний. На диаграмме возможно размещение трех типов состояний: начальное состояние *InitialState*, промежуточное состояние *State* и конечное состояние *FinalState*. Любое промежуточное состояние может иметь два действия: *EntryAction* – действие, которое будет выполнено сразу при входе в данное состояние, *ExitAction* – действие, которое будет выполнено при выходе из состояния. Начальное и конечное состояния имеют единственное действие. Для начального состояния это действие *InitialActivity*. Оно будет выполнено при

запуске автомата. Для конечного состояния это действие *FinalActivity*. Оно будет выполнено по завершению работы автомата в данном конечном состоянии.

Отношение *Transition* означает переход из одного состояния в другое. Каждый переход имеет название события *Trigger*, при возникновении которого будет сделан переход. Строка *Guard* представляет собой условие, выполнение которого необходимо для осуществления перехода. В случае наличия и выполнения необходимого условия, будет выполнено действие *Activity*. *Notation* – строка, которая генерируется автоматически и представляет собой полное описание перехода в формате *Trigger[Guard]/Action*.

3.2.2. Валидация диаграммы автомата

Для гарантии корректности проектируемой разработчиком диаграммы автомата, необходимо внести в нее механизм валидации. Следующие условия будут проверяться при проектировании диаграммы автомата:

1. Автомат должен иметь единственное начальное состояние.
2. Переходы из начального состояния не должны содержать событий.
3. Автомат должен содержать хотя бы одно конечное состояние (в случае невыполнения данного условия, разработчику будет выдано лишь предупреждение, а не ошибка).
4. Все переходы, кроме переходов из начального состояния, должны происходить по непустому событию.
5. Все состояния должны быть достижимы из начального.
6. Все состояния должны иметь уникальные имена.

3.3. Диаграмма объектов

Диаграмма объектов предназначена для визуального построения стартовой конфигурации объектов приложения. Для всех объектов, представленных на данной диаграмме, будет создан экземпляр объекта указанного типа. Также на этой диаграмме может быть задано отношение композиции между объектами. По полученной диаграмме может быть сгенерирован код для создания объектов, а

также отдельный метод, в котором будет запущен автомат для каждого объекта, имеющего автоматное поведение. В случае, если полученный метод должен быть стартовой точкой проекта, разработчик может указать это на диаграмме.

3.3.1. Метамодель диаграммы объектов

На рис. 12 представлена метамодель диаграммы объектов.

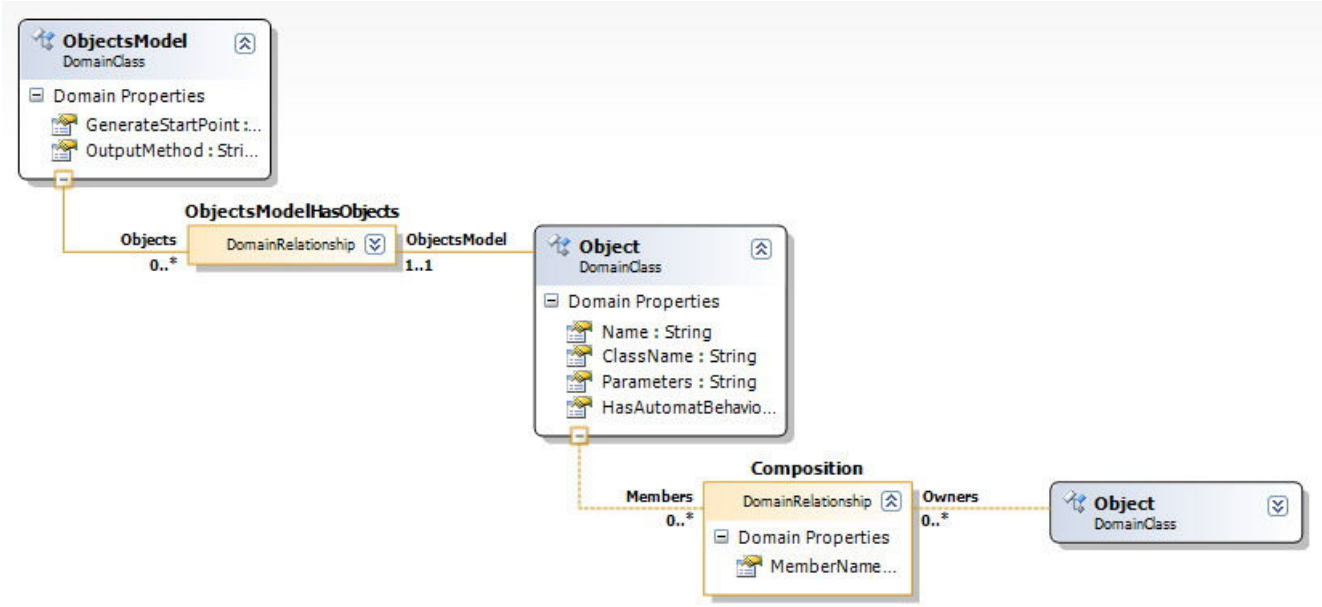

Рис. 12. Метамодель диаграммы объектов

Как и в предыдущих примерах, корневой элемент представляет саму диаграмму. В данном случае он имеет два свойства: *GenerateStartPoint* – поле, принимающее одно из двух значение *true* или *false*. Оно показывает, требуется ли автоматически генерировать код стартовой точки проекта, из которой запускается приложению. *OutputMethod* – поле, значение которого будет использовано в случае, если поле *GenerateStartPoint* принимает значение *false*. Отношение *ObjectsModelHasObjects* означает, что на модели будут располагаться объекты. Класс *Object* описывает объекты, которые могут быть добавлены на диаграмму. Каждый объект имеет имя, имя класса, а также параметры инициализации – *Name*, *ClassName* и *Parameters* соответственно.

Поле ***HasAutomatBehavior*** показывает, имеет ли конкретный объект автоматное поведение. Если конструируемый объект имеет автоматное

поведение, но автомат не должен быть запущен сразу при выполнении сгенерированного стартового метода, то разработчик может задать данному полю значение *false*. Отношение *Composition* означает, что один объект может являться полем или свойством другого объекта. При этом поле *MemberName* отношения задает имя инициализируемого поля или свойства.

3.3.2. Валидация диаграммы объектов

Для диаграммы объектов также должна быть введена процедура проверки правильности модели, представленной на диаграмме. Все объекты должны иметь уникальные имена. Также должно выполняться совпадение типов поля объекта и объекта, инициализирующего данное поле с помощью отношения композиции, представленного на диаграмме.

Метамодель диаграммы объектов не является полной. Она должна быть расширена для предоставления возможности инициализации всех полей и свойств объектов заданного типа. Пока такая возможность в работе отсутствует, и инициализация возможна только для полей и свойств, типом которых является любой класс.

3.4. Разработка шаблонов для генерации кода

При помощи инструментального средства *DSL Tools* возможна генерация кода по разработанной модели. Генерация кода осуществляется путем трансформации разработанного текстового шаблона для генерации кода определенной диаграммы (приложения 1–3). Для каждой из рассмотренных диаграмм разработан шаблон для генерации кода. Генерацию кода по шаблону можно осуществлять во время любого события изменения модели. Например, при сохранении редактируемой модели, шаблон будет автоматически трансформирован в исходный код. Полученный исходный код может быть скомпилирован и выполнен.

4. ОПИСАНИЕ РАЗРАБОТАННОГО ИНСТРУМЕНТАЛЬНОГО СРЕДСТВА

Разработанное инструментальное средство для визуального проектирования программ является плагином для среды разработки *Microsoft Visual Studio 2005*. После установки данного плагина расширения файлов "*.stateMachine*", "*.classes*" и "*.objects*" будут зарегистрированы как расширения файлов, содержащих диаграммы автомата, классов и объектов соответственно. В саму среду разработки *Microsoft Visual Studio 2005* будет встроена возможность создания новых языков. При добавлении в любой проект нового файла, можно будет выбрать *StateMachineLanguage*, *ClassesLanguage* или *ObjectsLanguage* (рис. 13).

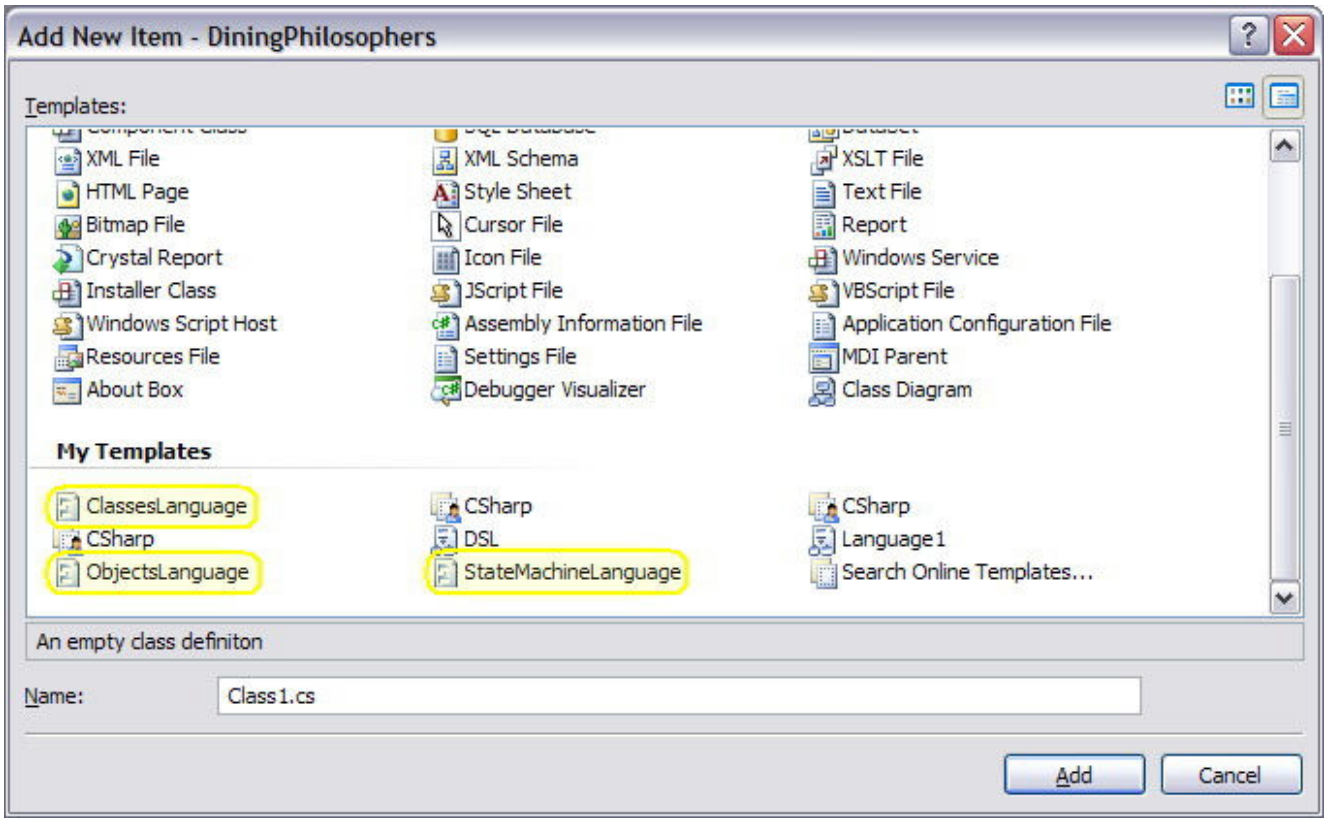

Рис. 13. Добавление нового файла к текущему проекту

На этом рисунке выделены новые типы файлов, которые были встроены в *Visual Studio* после установки разработанного инструментального средства. После добавления файлов нового типа в проект становится возможным создание диаграмм. По двойному щелчку мыши на созданном файле будет автоматически открываться визуальный редактор для изменения соответствующей модели.

Рассмотрим подробнее редактирование каждой из диаграмм.

4.1. Редактирование диаграммы классов

Для создания диаграммы классов, в проект необходимо добавить файл, которому соответствует тип *ClassesLanguage*, указанный в списке типов файлов и упомянутый выше. После добавления файла, его можно редактировать (рис. 14).

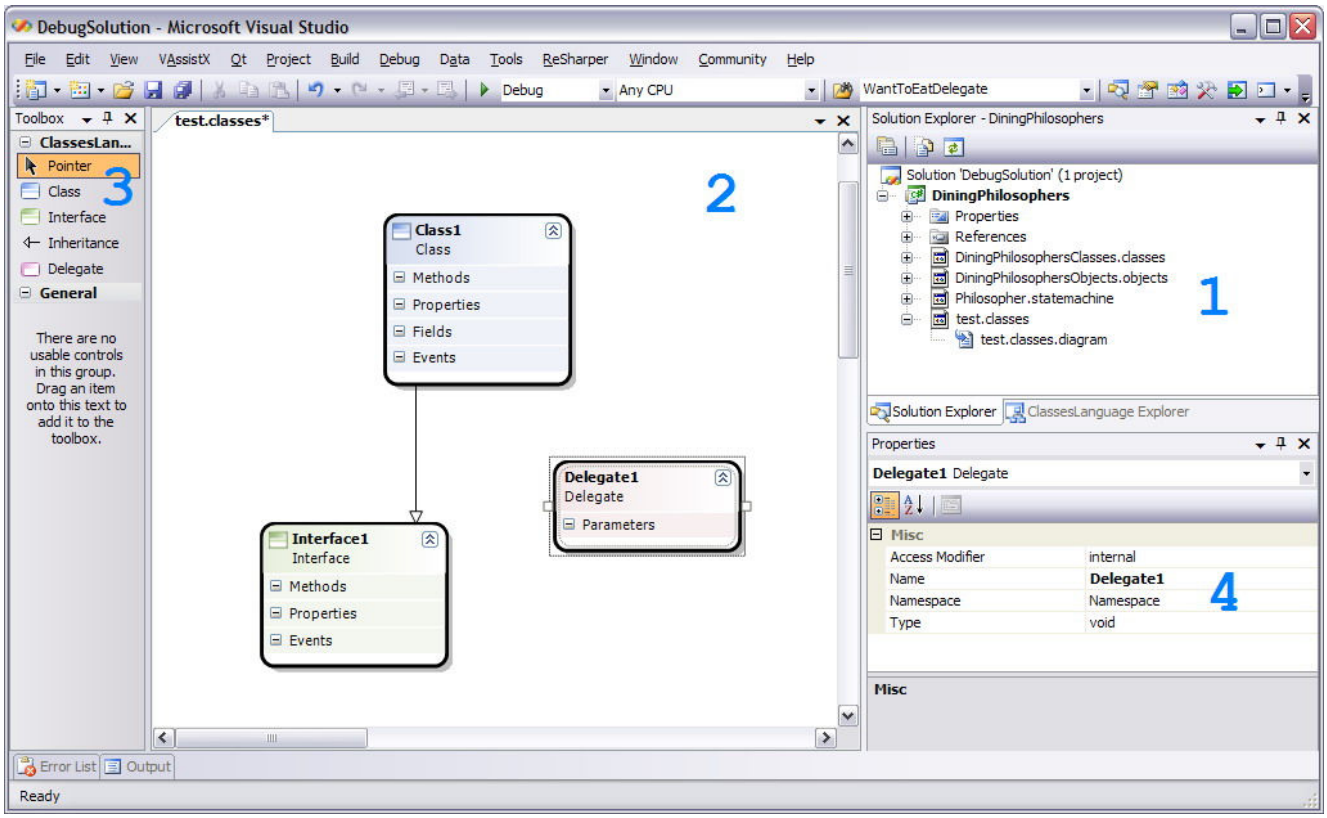

Рис. 14. Редактирование диаграммы классов в визуальном редакторе *VS2005*

На рис. 14 цифрами отмечены основные окна среды разработки во время редактирования диаграммы классов. Цифрами обозначены:

1. *Проводник Проекта* – древовидный список модулей и файлов проекта. Используя команду «*Add New Item…*» *Проводника Проекта*, возможно добавлять новые файлы, в том числе, и файлы созданных языков *StateMachineLanguage*, *ClassesLanguage* и *ObjectsLanguage*.

2. Редактируемый файл языка классов *.classes*. В данный файл визуально добавляются описания классов, интерфейсов, делегатов, а также отношения между классами и интерфейсами.
3. Инструментарий редактора языка классов – добавляет соответствующие графические представления в редактируемый файл.
4. Окно свойств активного объекта среды разработки.

Также, прямо на диаграмме классов, возможно добавление в конкретный класс автоматного поведения. Для этого по правому щелчку мыши на фигуре класса, требуется выбрать в контекстном меню пункт «*Add/Edit automat behavior*» (рис. 15). При этом автоматически будет создан файл с именем текущего класса и расширением *.stateMachine*. В случае, если класс уже имел автоматное поведение, в редакторе будет открыта уже созданная диаграмма автомата.

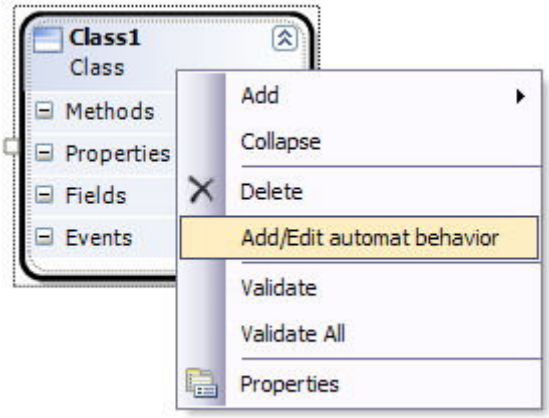

Рис. 15. Контекстное меню диаграммы при выделенной фигуре класса

Все классы, для которых добавлен автомат, имеют измененную иконку, на которой отображена буква «*A*» (рис. 16).

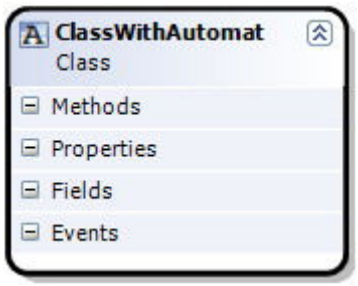

Рис. 16. Фигура класса, имеющего автоматное поведение

Реализация всех методов должна быть осуществлена непосредственно с диаграммы классов. Для этого требуется вызвать *Окно Свойств* для конкретного метода класса и отредактировать свойство *Inner Code* этого метода при помощи специального редактора кода *Диаграммы Классов* (рис. 17).

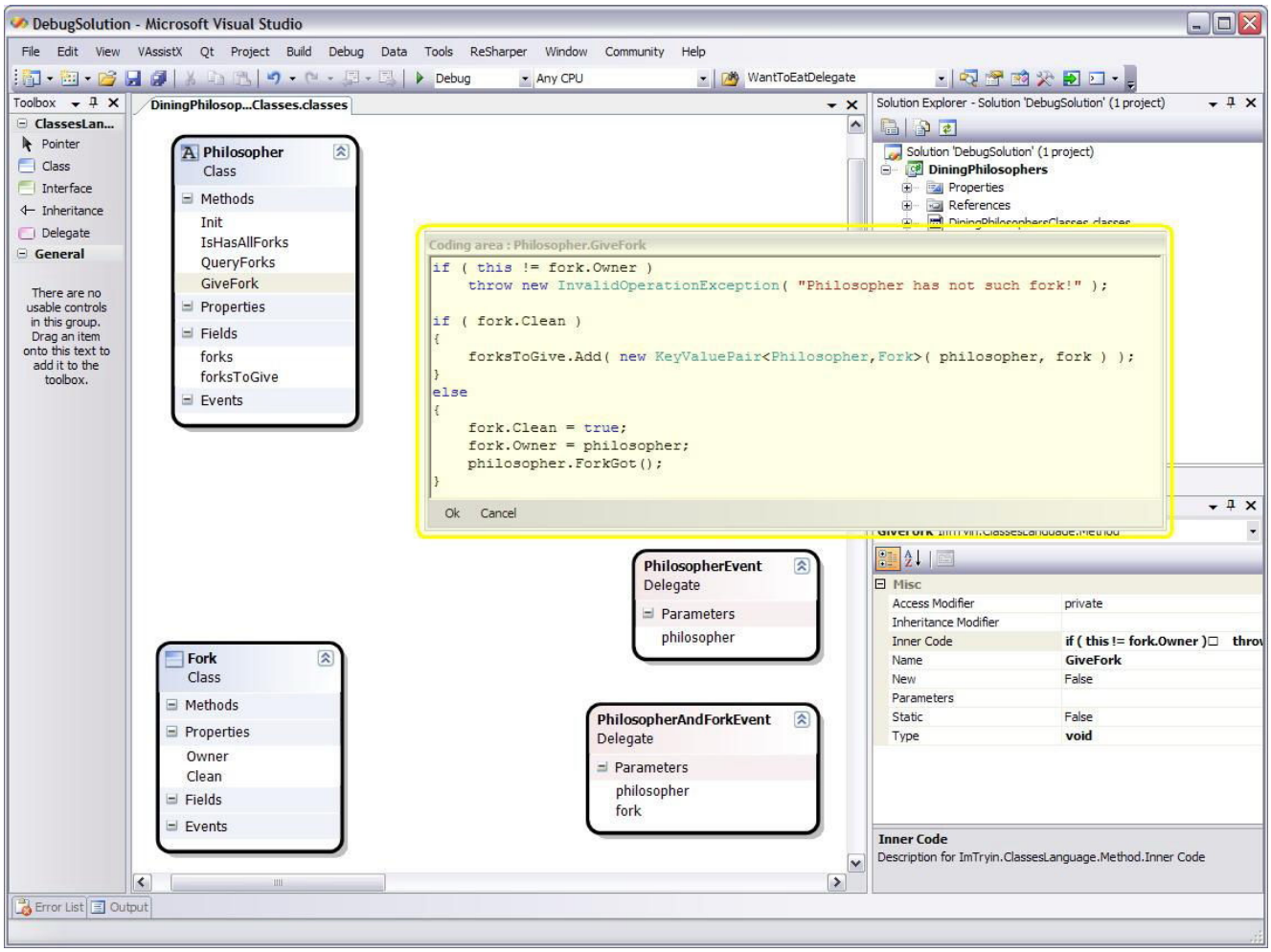

Рис. 17. Редактирование кода прямо из диаграммы классов

На рис. 17 выделен встроенный в диаграмму классов редактор кода в момент редактирования метода *GiveFork* класса *Philosopher*.

4.2. Редактирование диаграммы автомата

На рис. 18 отображена среда разработки во время редактирования диаграммы автомата. Возможно прямое создание диаграммы автомата методом добавления нового файла с помощью команды «*Add New Item..*», либо добавление автоматного поведения в класс, созданный при помощи диаграммы классов.

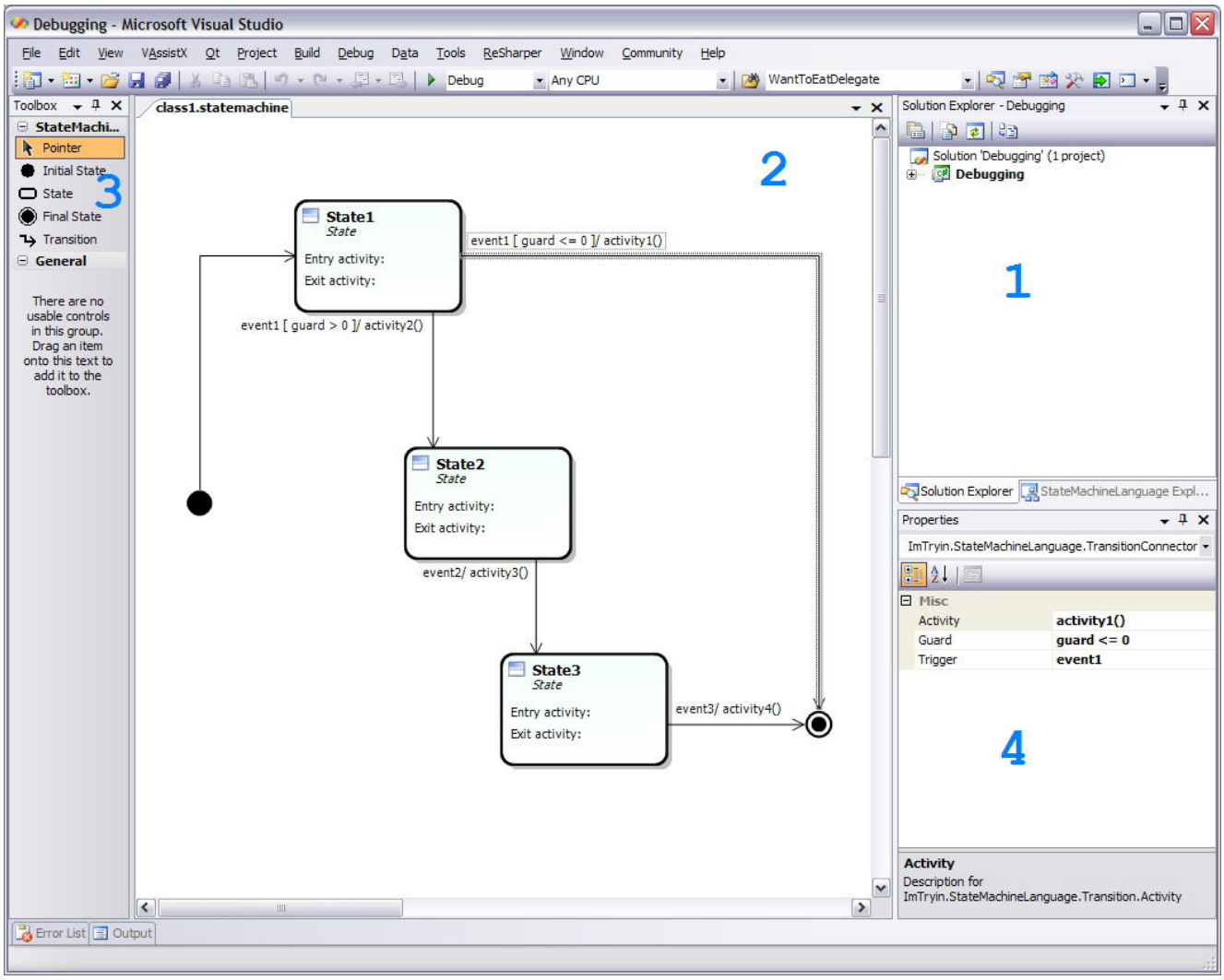

Рис. 18. Редактирование диаграммы автомата в визуальном редакторе *VS2005*

Цифрами обозначены аналогичные окна среды разработки, что и на рис. 14.

4.3. Редактирование диаграммы объектов

На рис. 19 отображена среда разработки во время редактирования диаграммы объектов.

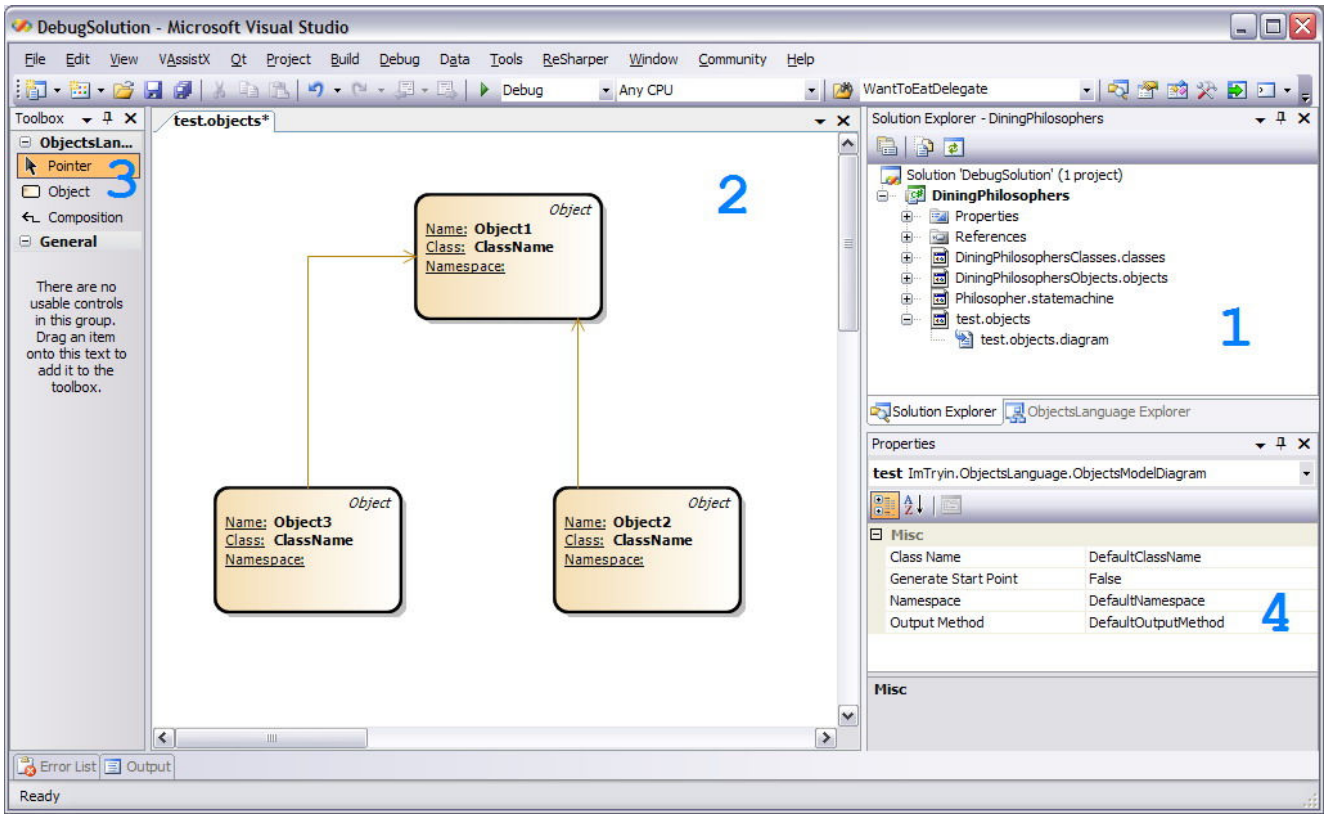

Рис. 19. Редактирование диаграммы объектов в визуальном редакторе *VS2005*

Обозначение окон среды разработки то же, что и на рис. 14. Для объектов, созданных на диаграмме, в случае если они являются объектом класса, для которого определен автомат, возможно указание автоматического запуска автомата для данного объекта. В случае, если для конкретного объекта установлена опция запуска автомата, объект будет иметь дополнительную иконку с буквой «*A*», как это показано на рис. 20.

Рис. 20. Фигура объекта, автомат которого будет автоматически запущен

5. ПРИМЕНЕНИЕ РАЗРАБОТАННОГО ИНСТРУМЕНТАЛЬНОГО СРЕДСТВА

В работе [4] была рассмотрена реализация задачи об обедающих философах [16] с использованием языка *StateMachineDesigner*. Для того, чтобы показать преимущества написания программ с помощью разрабатываемого инструмента, реализуем ту же задачу, но с использованием всех трех разработанных диаграмм.

Процесс разработки любого приложения с помощью разработанного средства делится на четыре этапа:

1. Проектирование классов, интерфейсов и создание делегатов с помощью *Диаграммы Классов*.
2. Добавление к некоторым классам автоматного поведения и проектирование автоматов с помощью *Диаграммы Автомата*.
3. Реализация методов классов путем редактирования кода в специальном редакторе, вызываемым из *Диаграммы Классов*.
4. Создание начальной конфигурации экземпляров объектов и связей между ними с помощью *Диаграммы Объектов*.

Используя описанную методику, была реализована задача об обедающих философах. На рис. 21 изображена *Диаграмма Классов* проекта.

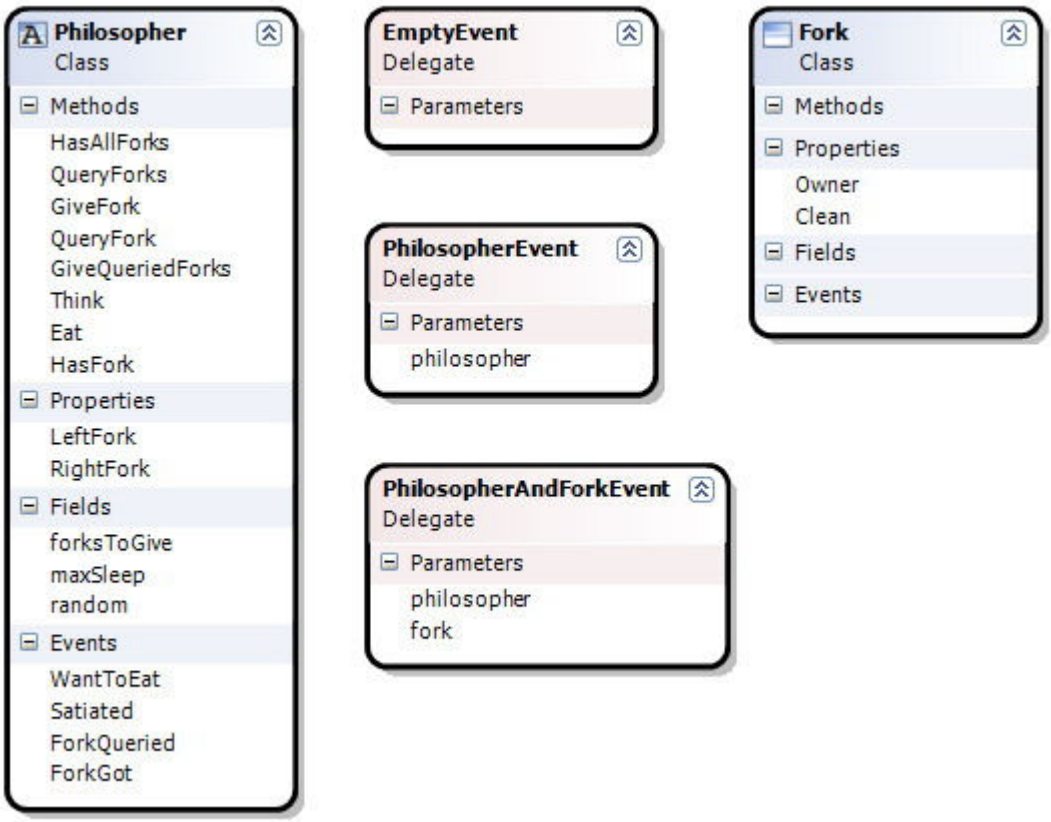

Рис. 21. Диаграмма классов для решения задачи об обедающих философах

На диаграмме отображены классы *Philosopher* и *Fork*, обозначающие классы философа и вилки соответственно. Также на диаграмме отображены три делегата *EmptyEvent*, *PhilosopherEvent*, *PhilosopherAndForkEvent* – типы сообщений, которые будут использоваться автоматом. Класс философ имеет автоматное поведение, поэтому на его иконке отображена буква «*A*».

В качестве автомата для философа был использован упрощенный автомат из трех состояний. Автомат приведен на рис. 22.

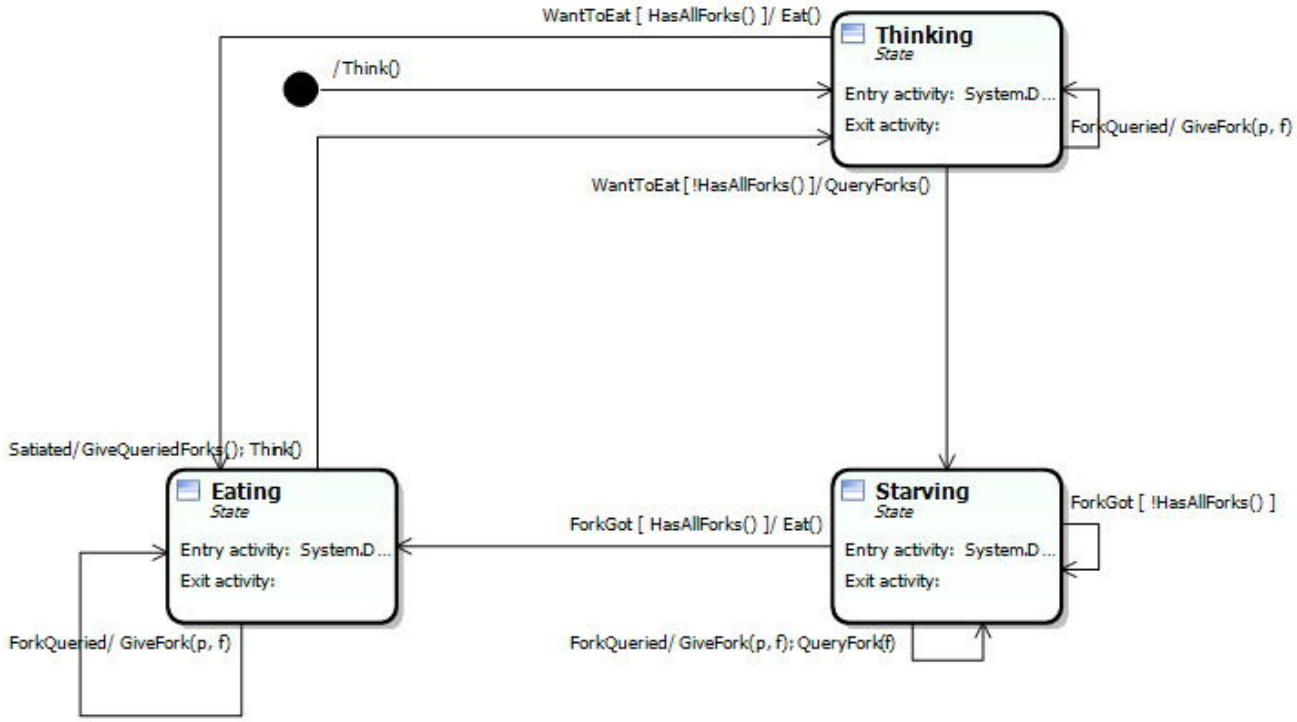

Рис. 22. Диаграмма автомата для класса *Philosopher*

На диаграмме автомата отображены три основных состояния философа – «Думаю», «Ем» и «Голодаю». Для каждого из состояний установлено действие – вывод во вспомогательное окно среды разработки *Visual Studio 2005* соответствующего сообщение вида «*StateMachineTrace: philosopher [x] is y…*». В этом сообщении вместо символа «*x*» отображается уникальный номер объекта философа, автомат которого осуществил переход, а вместо символа «*y*» одно из слов «*thinking*», «*starving*» или «*eating*», если философ думает, голодает или ест соответственно.

После того, как построена диаграмма классов, диаграмма автомата и написаны методы диаграммы классов, остается последний шаг – создание диаграммы объектов для обозначения начального состояния системы. На рис. 23 отображена диаграмма объектов.

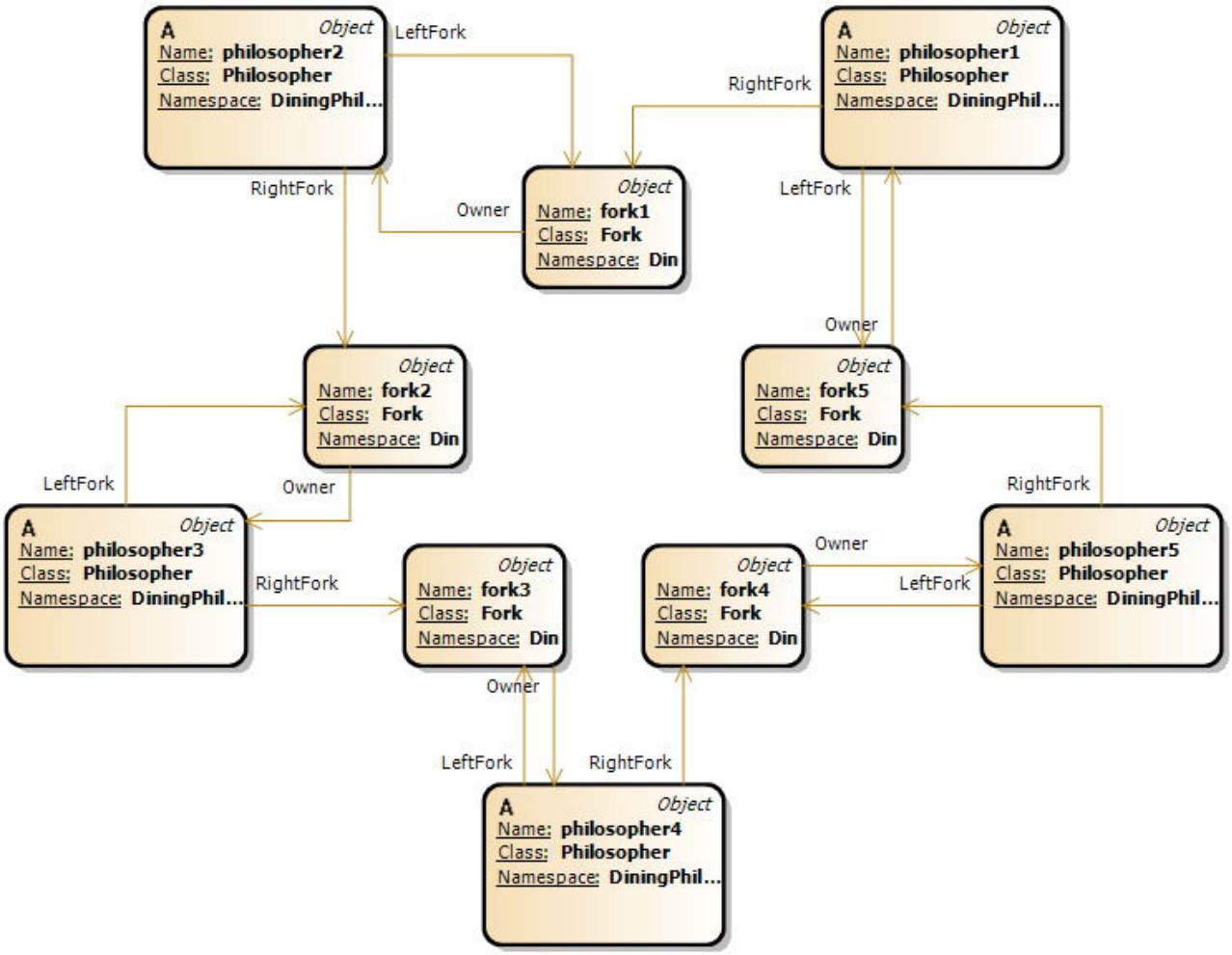

Рис. 23. Диаграмма объектов для задачи об обедающих философах

На диаграмме объектов отображены пять философов и пять вилок. Стрелки, которые ведут от философа к вилке означают, что философ может владеть данной вилкой. Подписи стрелок, которые направлены от объектов класса *Philosopher* к объектам класса *Fork*, *LeftFork* и *RightFork* задают имя поля класса *Philosopher*. Оно будет проинициализировано конкретным объектом класса *Fork*. Подпись *Owner* на стрелках, которые ведут от объектов класса *Fork* к объектам класса *Philosopher*, означает какой философ будет начальным владельцем вилки.

После того, как все диаграммы спроектированы, остается только указать в свойствах для *Диаграммы Объектов* имя генерируемого класса, области видимости имен, а также указать, что точка входа в проект должна быть сгенерирована автоматически, присвоив полю «*Generate Start Point*» значение *true*.

При сохранении каждой из трех диаграмм, автоматически будет сгенерирован исходный код на языке *C#*. После генерации кода для всех трех диаграмм, можно запускать полученное приложение. Ниже приведена часть протокола одновременной работы пяти автоматов:

```
StateMachineTrace: Philosopher [14] is starving...
StateMachineTrace: Philosopher [13] is eating...
StateMachineTrace: Philosopher [15] is thinking...
StateMachineTrace: Philosopher [15] is thinking...
StateMachineTrace: Philosopher [11] is starving...
StateMachineTrace: Philosopher [15] is starving...
StateMachineTrace: Philosopher [15] is starving...
StateMachineTrace: Philosopher [15] is starving...
StateMachineTrace: Philosopher [12] is eating...
StateMachineTrace: Philosopher [11] is starving...
StateMachineTrace: Philosopher [14] is eating...
StateMachineTrace: Philosopher [14] is eating...
StateMachineTrace: Philosopher [13] is thinking...
StateMachineTrace: Philosopher [13] is starving...
StateMachineTrace: Philosopher [15] is eating...
StateMachineTrace: Philosopher [14] is thinking...
StateMachineTrace: Philosopher [14] is starving...
StateMachineTrace: Philosopher [13] is starving...
StateMachineTrace: Philosopher [13] is starving...
StateMachineTrace: Philosopher [14] is starving...
StateMachineTrace: Philosopher [12] is thinking...
StateMachineTrace: Philosopher [12] is thinking...
StateMachineTrace: Philosopher [12] is starving...
StateMachineTrace: Philosopher [15] is thinking...
StateMachineTrace: Philosopher [14] is starving...
StateMachineTrace: Philosopher [15] is thinking...
StateMachineTrace: Philosopher [14] is starving...
StateMachineTrace: Philosopher [15] is starving...
StateMachineTrace: Philosopher [11] is eating...
StateMachineTrace: Philosopher [12] is starving...
StateMachineTrace: Philosopher [11] is eating...
StateMachineTrace: Philosopher [15] is starving...
StateMachineTrace: Philosopher [11] is eating...
StateMachineTrace: Philosopher [13] is eating...
StateMachineTrace: Philosopher [13] is eating...
StateMachineTrace: Philosopher [13] is thinking...
```

При переходе каждого автомата из одного состояния в другое, выводится сообщение в вышеупомянутом формате. Сообщение содержит уникальный номер философа, а также имя состояния автомата, в которое был осуществлен переход. По приведенному протоколу можно видеть, что каждый из философов попеременно находится в состоянии «*Eating*», «*Thinking*» и «*Starving*», что говорит о правильной работе автоматов.

Сгенерированный по диаграммам исходный код приведен в приложениях 4–6.

ЗАКЛЮЧЕНИЕ

В результате выполненной работы создано инструментальное средство визуального проектирования программ в среде разработки программного обеспечения *Microsoft Visual Studio 2005*. С его помощью процесс написания программ становится более структурированным, а наличие основных диаграмм с возможностью генерации по ним исходных текстов, существенно уменьшает объем написанного кода.

Инструментальное средство может быть использовано при разработке проектов различной направленности, но будет особенно полезно при создании реактивных систем. При этом с помощью реализованных диаграмм одновременно отображаются статическая и динамическая модели приложения, что до настоящего времени отсутствовало в программах-аналогах, за исключением инструментального средства *Unimod*.

Разработанное средство позволяет выполнять не только разработку проектов, но и дополнение существующего кода. Разработка классов с помощью *Диаграммы Классов* не только упрощает их проектирование, но и позволяет синхронизировать исходный код и диаграмму. Возможность визуального создания объектов с помощью *Диаграммы Объектов* избавляет разработчика от написания кода простой инициализации объектов, а также кода стартовой точки приложения. В случае создания системы из нескольких автоматов, которые должны быть одновременно запущены в самом начале, либо в случае запуска одного автомата другим, каждый автомат будет по умолчанию создаваться в новом потоке и работать параллельно с уже запущенными автоматами.

В заключение отметим, что данное средство позволяет разрабатывать автоматные программы визуальным способом и полноценно расширяет возможности среды разработки *Microsoft Visual Studio 2005*. Средство

обеспечивает построение трех типов диаграмм (диаграммы классов, диаграммы автомата, диаграммы объектов), на основе которых генерируется соответствующий исходный код. Для построения программы в целом функции входных и выходных воздействий пишутся вручную.

Для установки приложения загрузите архив с инсталляционным пакетом, распакуйте его и запустите setup.exe для каждого из трех типов диаграмм: *ClassesLanguage*, *ObjectsLanguage*, *StateMachineLanguage*. При этом дополнительные компоненты будут встроены в *Visual Studio 2005*. Для работы приложения необходимо иметь на компьютере установленную версию *Visual Studio 2005*, а также *Visual Studio 2005 SDK 2.0 - February 2007 RTM*.

Использование средства продемонстрировано на примере реализации задачи об обедающих философах.

Ведутся работы по созданию следующей версии средства, позволяющего использовать вложенные автоматы.

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. *Ларионов А.* Визуальный язык автоматного программирования для Microsoft Visual Studio 2005. СПбГУ ИТМО. 2006. http://is.ifmo.ru/papers/larionov/
5. *Троелсен Э.* C# и платформа .Net. Библиотека программиста. СПб.: Питер, 2003.
6. *Rational Rose.* Market-leading modeling environment for the entire team. http://www-304.ibm.com/jct03001c/software/awdtools/developer/rose/
7. *Sybase* PowerDesigner. http://www.sybase.com/products/modelingmetadata/powerdesigner/
8. *Sparx Systems.* Enterprise Architect Professional. http://www.sparxsystems.com/
9. *ApeSoft* SmartState. http://www.smartstatestudio.com/
10. *Microsoft* Windows Workflow Foundation. http://wf.netfx3.com/
11. *Visual Studio Code Name "Orcas".* http://msdn2.microsoft.com/en-us/vstudio/aa700830.aspx
12. *Unimod*. http://unimod.sourceforge.net/
13. *Microsoft Visual Studio Developer Center.* Domain-Specific Language Tools. http://msdn.microsoft.com/vstudio/DSLTools/
14. *Borland* Together. http://www.borland.com/us/products/together/

15. *Шалыто А. А., Туккель Н. И.* Программирование с явным выделением состояний //Мир ПК. 2001. № 8, с. 116–121; № 9, с. 132–138. http://is.ifmo.ru/works/mirk/
16. *Wikipedia, the free encyclopedia.* Dining philosophers problem. http://en.wikipedia.org/wiki/Dining_philosophers_problem

ПРИЛОЖЕНИЯ

Приложение 1. Шаблон генерации кода по диаграмме классов

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation"
debug="true" #>
<#@ output extension=".cs" #>
<#@ ClassesLanguage processor="ClassesLanguageDirectiveProcessor"
requires="fileName='%FILENAMEMARKER%'" #>
<#@ import namespace="System.Diagnostics" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Text.RegularExpressions" #>

<#
foreach ( ImTryin.ClassesLanguage.Type typeArtefact in ClassesModel.Types )
{
#>
namespace <#= typeArtefact.Namespace #>
{
<#
    if ( typeArtefact is Interface )
        OutputInterfaceDeclaration( (Interface)typeArtefact );
    else if ( typeArtefact is Class )
        OutputClassDeclaration( (Class)typeArtefact );
    else if ( typeArtefact is ImTryin.ClassesLanguage.Delegate )
        OutputDelegateDeclaration( (ImTryin.ClassesLanguage.Delegate)typeArtefact
);
    else
        Debug.Fail("Some kind of unsupported TypeArtefact. Can not generate
code.");
#>
}

<#
}
#>
<#+
private void OutputDelegateDeclaration(ImTryin.ClassesLanguage.Delegate
delegateArtefact)
{
    string parametersString = string.Empty;
    foreach (Parameter param in delegateArtefact.Parameters)
    {
        if (parametersString.Length > 0)
            parametersString += ", ";
        parametersString += param.Type + " " + param.Name;
    }
#>
    <#= delegateArtefact.AccessModifier #> delegate <#= delegateArtefact.Type #>
<#= delegateArtefact.Name #>(<#= parametersString #>);
<#+
}

private void OutputInterfaceDeclaration(Interface interfaceArtefact)
{
#>
```

```
    <#= interfaceArtefact.AccessModifier #> interface <#= interfaceArtefact.Name
#> <#= GetParentClassesString(interfaceArtefact) #>
    {
<#+
    foreach (ImTryin.ClassesLanguage.Event eventArtefact in
interfaceArtefact.Events)
        OutputEventDeclaration(eventArtefact, interfaceArtefact);
    foreach (Property property in interfaceArtefact.Properties)
        OutputPropertyDeclaration(property, interfaceArtefact);
    foreach (Method method in interfaceArtefact.Methods)
        OutputMethodDeclaration(method, interfaceArtefact);
#>
    }
<#+
}

private void OutputClassDeclaration(Class classArtefact)
{
#>
    <#= classArtefact.AccessModifier #> <#= classArtefact.InheritanceModifier #>
partial class <#= classArtefact.Name #> <#= GetParentClassesString(classArtefact)
#>
    {
<#+
    foreach (Field field in classArtefact.Fields)
        OutputFieldDeclaration(field, classArtefact);
    foreach (ImTryin.ClassesLanguage.Event eventArtefact in classArtefact.Events)
        OutputEventDeclaration(eventArtefact, classArtefact);
    foreach (Property property in classArtefact.Properties)
        OutputPropertyDeclaration(property, classArtefact);
    foreach (Method method in classArtefact.Methods)
        OutputMethodDeclaration(method, classArtefact);
#>
    }
<#+
}

private void OutputFieldDeclaration(Field field, ImTryin.ClassesLanguage.Type
typeArtefact)
{
    Class classArtefact = typeArtefact as Class;
    string classStaticModifier = ((classArtefact != null) &&
(classArtefact.InheritanceModifier == "static")) ? "static" : "";
#>

        <#= field.AccessModifier #> <#= (field.Static && (classStaticModifier ==
"")) ? "static" : "" #> <#= (field.ConstantKind) ? "const" : "" #> <#= field.Type
#> <#= field.Name #> <#= (field.InitialValue.Length > 0) ? "= " +
field.InitialValue : "" #>;
<#+
}

private void OutputPropertyDeclaration(Property property,
ImTryin.ClassesLanguage.Type typeArtefact)
{
    if (typeArtefact is Interface)
    {
#>

        <#= (property.New) ? "new" : "" #> <#= property.Type #> <#= property.Name
#>
```

```
        {
            get;
            set;
        }
<#+
    }
    else
    {
#>

        private <#= property.Type #> <#= GetMappedUniqueName(property.Name) #> =
<#= property.InitialValue #>;

        <#= property.AccessModifier #> <#= (property.New) ? "new" : "" #> <#=
property.Type #> <#= property.Name #>
        {
            get { return <#= GetMappedUniqueName(property.Name) #>; }
            set { <#= GetMappedUniqueName(property.Name) #> = value; }
        }
<#+
    }
}

private void OutputMethodDeclaration(Method method, ImTryin.ClassesLanguage.Type
typeArtefact)
{
#>

        <#= (typeArtefact is Interface) ? "" : method.AccessModifier #> <#=
(method.New) ? "new" : "" #> <#= method.Type #> <#= method.Name #>( <#=
method.Parameters #> )
        <#= (typeArtefact is Interface) ? ";" : "{\r\n            " +
method.InnerCode + "\r\n        }" #>
<#+
}

private void OutputEventDeclaration(ImTryin.ClassesLanguage.Event eventArtefact,
ImTryin.ClassesLanguage.Type typeArtefact)
{
#>

        <#= (typeArtefact is Interface) ? "" : eventArtefact.AccessModifier #> <#=
(eventArtefact.New) ? "new" : "" #> event <#= eventArtefact.Type #> <#=
eventArtefact.Name #>;
<#+
}

private string GetParentClassesString(ImTryin.ClassesLanguage.Type type)
{
    string list = "";
    string baseClass = "";
    foreach (ImTryin.ClassesLanguage.Type baseType in type.BaseTypes)
    {
        if (baseType is Interface)
        {
            if (list.Length > 0)
                list += ", ";
            list += baseType.Namespace + "." + baseType.Name;
        }
        else if (baseType is Class)
            baseClass = baseType.Namespace + "." + baseType.Name;
```

```
            else
                Debug.Fail("Base class is not a class or an interface.");
        }
        if (baseClass.Length > 0)
        {
            if (list.Length > 0)
                list = ", " + list;
            list = baseClass + list;
        }
        if (list.Length > 0)
            list = " : " + list;
        return list;
    }

        private Dictionary<string, string> _uniqueNames = null;

        private string GetMappedUniqueName(string name)
        {
            if (_uniqueNames == null)
                _uniqueNames = new Dictionary<string, string>();
            if (!_uniqueNames.ContainsKey(name))
                _uniqueNames[name] = GetSingleWord(name + "_" +
Guid.NewGuid().ToString(), string.Empty);
                //_uniqueNames[name] = GetSingleWord(name, string.Empty);
            return _uniqueNames[name];
        }

            private string GetSingleWord(string name, string replacer)
        {
            return Regex.Replace( name, "[^A-Za-z_0-9]", replacer );
        }
          #>
```

Приложение 2. Шаблон генерации кода по диаграмме автомата

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation"
debug="true"#>
<#@ output extension=".cs" #>
<#@ StateMachineLanguage processor="StateMachineLanguageDirectiveProcessor"
requires="fileName='%FILENAMEMARKER%'" #>
<#@ import namespace="System.Diagnostics" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Text.RegularExpressions" #>

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

namespace <#= StateMachine.Namespace #>
{
    public partial class <#= StateMachine.ClassName #>
    {
        private Queue< KeyValuePair< <#= GetMappedUniqueName("EventName") #>,
object[] > > <#= GetMappedUniqueName("eventQueue") #> = new Queue< KeyValuePair<
<#= GetMappedUniqueName("EventName") #>, object[] > >();

        private enum <#= GetMappedUniqueName("StateName") #>
        {
            Initial,
            Final,
<#
List<State> states = GetStates();
for (int i = 0; i < states.Count; i++)
{
#>
            <#= GetMappedUniqueName(states[i].Name) + ((i + 1 == states.Count) ?
"" : ",") #>
<#
}
#>
        }

        private enum <#= GetMappedUniqueName("EventName") #>
        {
<#
Dictionary<string, List<Transition>> eventTransitionsMap =
GetEventTransitionsMap();

List<string> eventNames = new List<string>();
foreach (string e in eventTransitionsMap.Keys)
{
    if (e != string.Empty)
        eventNames.Add(e);
}

for (int i = 0; i < eventNames.Count; i++)
{
#>
            <#= GetMappedUniqueName( GetEventName(eventNames[i]) ) + ((i + 1 ==
eventNames.Count) ? "" : ",") #>
<#
}
```

```
#>
        }

        private <#= GetMappedUniqueName("StateName") #> <#=
GetMappedUniqueName("currentState") #> = <#= GetMappedUniqueName("StateName")
#>.Initial;

        public void StartStateMachine()
        {
            if (<#= GetMappedUniqueName("currentState") #> != <#=
GetMappedUniqueName("StateName") #>.Initial)
                throw new InvalidOperationException("State machine is already
started!");

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

            <#= GetMappedUniqueName("Initialize") #>();

            Thread thread = new Thread(<#= GetMappedUniqueName("StateMachineLoop")
#>);
            thread.Start();
        }

        private void <#= GetMappedUniqueName("Initialize") #>()
        {
<#
InitialState initialState = GetInitialState();
if ((initialState != null) && (initialState.InitialActivity.Length > 0))
{
#>
            <#= initialState.InitialActivity #>;
<#
}

Dictionary<BaseState, List<Transition>> transitionsByStateMap =
GroupTransitionsByStates(GetTransitions());
List<Transition> initialTransitions = (initialState == null) ? new
List<Transition>() : transitionsByStateMap[initialState];
bool elseStatement = false;
if (initialTransitions != null)
{
    foreach (Transition transition in initialTransitions)
    {
        if (transition.Guard.Length > 0)
        {
            if (elseStatement)
            {
#>
            else if (<#= transition.Guard #>)
<#
            }
            else
            {
#>
            if (<#= transition.Guard #>)
<#
                elseStatement = true;
            }
        }
#>
            {
```

```
<#
        if (transition.Activity.Length > 0)
        {
#>
                <#= transition.Activity #>;
<#
        }
        State firstState = transition.Target as State;
        if (firstState != null)
        {
            if (firstState.EntryActivity.Length > 0)
            {
#>
                <#= firstState.EntryActivity #>;
<#
            }
#>
                <#= GetMappedUniqueName("SetNewState") #>(<#=
GetMappedUniqueName("StateName") #>.<#=
GetMappedUniqueName(transition.Target.Name) #>);
<#
        }
        FinalState immFinalState = transition.Target as FinalState;
        if (immFinalState != null)
        {
            if (immFinalState.FinalActivity.Length > 0)
            {
#>
                <#= immFinalState.FinalActivity #>;
<#
            }
#>
                <#= GetMappedUniqueName("SetNewState") #>(<#=
GetMappedUniqueName("StateName") #>.Final);
<#
        }
#>
            }
<#
    }
}
#>
        }

        private void <#= GetMappedUniqueName("Finalize") #>()
        {
                <#= GetMappedUniqueName("UnsubscribeEvents") #>();
        }

        private void <#= GetMappedUniqueName("StateMachineLoop") #>()
        {
            while (true)
            {
                KeyValuePair< <#= GetMappedUniqueName("EventName") #>, object[] >
eventToProcess;
                lock (<#= GetMappedUniqueName("eventQueue") #>)
                {
                    while (<#= GetMappedUniqueName("eventQueue") #>.Count == 0)
                        Monitor.Wait(<#= GetMappedUniqueName("eventQueue") #>);
                    eventToProcess = <#= GetMappedUniqueName("eventQueue")
#>.Dequeue();
```

```
                }
                switch (eventToProcess.Key)
                {
<#
foreach (string eventName in eventNames)
{
    string parametersString = string.Empty;
    List< KeyValuePair<string, string> > parameters =
GetParametersPairs(eventName);
    for (int i = 0; i < parameters.Count; i++)
    {
        KeyValuePair<string, string> pair = parameters[i];
        if (parametersString.Length > 0)
            parametersString += ", ";
        parametersString += "(" + pair.Key + ")eventToProcess.Value[" +
i.ToString() + "]";
    }
#>
                    case <#= GetMappedUniqueName("EventName") #>.<#=
GetMappedUniqueName( GetEventName(eventName) ) #> :
                        <#= GetMappedUniqueName( GetEventName(eventName) ) #>(<#=
parametersString #>);
                        break;
<#
}
#>
                    default:
                        break;
                }
                if (<#= GetMappedUniqueName("currentState") #> == <#=
GetMappedUniqueName("StateName") #>.Final)
                    break;
            }
        }

        private void <#= GetMappedUniqueName("SetNewState") #>(<#=
GetMappedUniqueName("StateName") #> nextState)
        {
            if (nextState == <#= GetMappedUniqueName("StateName") #>.Final)
                <#= GetMappedUniqueName("Finalize") #>();
            else
                <#= GetMappedUniqueName("currentState") #> = nextState;
        }

<#
eventTransitionsMap = GetEventTransitionsMap();
foreach (string trigger in eventTransitionsMap.Keys)
{
    if (trigger.Length > 0)
    {
#>
        private void On<#= GetMappedUniqueName( GetEventName(trigger) ) #>(<#=
GetParametersString(trigger, false) #>)
        {
            lock (<#= GetMappedUniqueName("eventQueue") #>)
            {
                KeyValuePair< <#= GetMappedUniqueName("EventName") #>, object[] >
pair = new KeyValuePair< <#= GetMappedUniqueName("EventName") #>, object[] >
                    (
                        <#= GetMappedUniqueName("EventName") #>.<#=
GetMappedUniqueName( GetEventName(trigger) ) #>,
```

```
                        new object[] { <#= GetParametersString(trigger, true) #> }
                    );
                <#= GetMappedUniqueName("eventQueue") #>.Enqueue(pair);
                Monitor.PulseAll(<#= GetMappedUniqueName("eventQueue") #>);
            }
        }

        private void <#= GetMappedUniqueName( GetEventName(trigger) ) #>(<#=
GetParametersString(trigger, false) #>)
        {
            switch (<#= GetMappedUniqueName("currentState") #>)
            {
<#
        List<Transition> transitionsByCurrentEvent = eventTransitionsMap[trigger];
        Dictionary<BaseState, List<Transition>> transitionsByState =
GroupTransitionsByStates(transitionsByCurrentEvent);
        foreach (BaseState state in transitionsByState.Keys)
        {
#>
                case <#= GetMappedUniqueName("StateName") #>.<#=
GetMappedUniqueName(state.Name) #> :
                    {
<#
            List<Transition> transitionsFromCurrentState =
transitionsByState[state];
            elseStatement = false;
            foreach (Transition transition in transitionsFromCurrentState)
            {
                if (transition.Guard.Length > 0)
                {
                    if (elseStatement)
                    {
#>
                        else if (<#= transition.Guard #>)
<#
                    }
                    else
                    {
#>
                        if (<#= transition.Guard #>)
<#
                        elseStatement = true;
                    }
                }
#>
                        {
<#
                State sourceState = state as State;
                if ((sourceState != null) && (sourceState.ExitActivity.Length >
0))
                {
#>
                            <#= sourceState.ExitActivity #>;
<#
                }
                if (transition.Activity.Length > 0)
                {
#>
                            <#= transition.Activity #>;
<#
                }
```

```
                    State targetState = transition.Target as State;
                    if (targetState != null)
                    {
                        if (targetState.EntryActivity.Length > 0)
                        {
#>
                                <#= targetState.EntryActivity #>;
<#
                        }
#>
                                <#= GetMappedUniqueName("SetNewState") #>(<#=
GetMappedUniqueName("StateName") #>.<#=
GetMappedUniqueName(transition.Target.Name) #>);
<#
                    }
                    FinalState finalState = transition.Target as FinalState;
                    if (finalState != null)
                    {
                        if (finalState.FinalActivity.Length > 0)
                        {
#>
                                <#= finalState.FinalActivity #>;
<#
                        }
#>
                                <#= GetMappedUniqueName("SetNewState") #>(<#=
GetMappedUniqueName("StateName") #>.Final);
<#
                    }
#>
                            }
<#
                }
#>
                            break;
                        }
<#
            }
#>
                    default:
                        break;
                }
            }
<#
        }
}
#>

        private void <#= GetMappedUniqueName("SubscribeEvents") #>()
        {
<#
foreach (string trigger in eventTransitionsMap.Keys)
{
    if (trigger.Length > 0)
    {
#>
            <#= GetEventName(trigger) #> += On<#= GetMappedUniqueName(
GetEventName(trigger) ) #>;
<#
    }
    else
```

```
    {
        List<Transition> emptyTriggerTransitions = eventTransitionsMap[trigger];
        foreach (Transition transition in emptyTriggerTransitions)
        {
            if (!(transition.Source is InitialState))
            {
                Debug.Fail("Some of the transitions have no events!");
                break;
            }
        }
    }
}
#>
        }

        private void <#= GetMappedUniqueName("UnsubscribeEvents") #>()
        {
<#
foreach (string trigger in eventTransitionsMap.Keys)
{
    if (trigger.Length > 0)
    {
#>
            <#= GetEventName(trigger) #> -= On<#= GetMappedUniqueName(
GetEventName(trigger) ) #>;
<#
    }
}
#>
        }
    }
}

<#+
    private InitialState _initialState = null;
    private List<State> _states = null;
    private List<FinalState> _finalStates = null;
    private List<Transition> _transitions = null;
    private Dictionary<string, List<Transition>> _eventTransitionsMap = null;
    private Dictionary<string, string> _uniqueNames = null;

    private List<State> GetStates()
    {
        if (_states == null)
        {
            _states = new List<State>();
            foreach (BaseState baseState in StateMachine.States)
            {
                if (baseState is State)
                {
                    _states.Add((State)baseState);
                }
            }
        }
        return _states;
    }

    private InitialState GetInitialState()
    {
        if (_initialState == null)
        {
```

```
            foreach (BaseState baseState in StateMachine.States)
            {
                if (baseState is InitialState)
                {
                    if (_initialState != null)
                        Debug.Fail("Automat has more than one initial state!");
                    _initialState = (InitialState)baseState;
                }
            }
        }
        if (_initialState == null)
            Debug.Fail("Automat has no initial state!");
        return _initialState;
    }

    private List<FinalState> GetFinalStates()
    {
        foreach (BaseState baseState in StateMachine.States)
        {
            if (baseState is FinalState)
            {
                if (_finalStates == null)
                    _finalStates = new List<FinalState>();
                _finalStates.Add((FinalState)baseState);
            }
        }
        if ((_finalStates == null) || (_finalStates.Count == 0))
            Debug.Fail("Automat has no final states!");
        return _finalStates;
    }

    private List<Transition> GetTransitions()
    {
        if (_transitions == null)
        {
            _transitions = new List<Transition>();
            foreach (BaseState sourceState in StateMachine.States)
            {
                foreach (BaseState targetState in StateMachine.States)
                {
                    foreach (Transition transition in
Transition.GetLinks(sourceState, targetState))
                        _transitions.Add(transition);
                }
            }
        }
        return _transitions;
    }

    private Dictionary<string, List<Transition>> GetEventTransitionsMap()
    {
        List<Transition> transitions = GetTransitions();
        if (_eventTransitionsMap == null)
        {
            _eventTransitionsMap = new Dictionary<string, List<Transition>>();
            foreach (Transition transition in transitions)
            {
                if (!_eventTransitionsMap.ContainsKey(transition.Trigger))
                    _eventTransitionsMap[transition.Trigger] = new
List<Transition>();
                _eventTransitionsMap[transition.Trigger].Add(transition);
```

```
            }
        }
        return _eventTransitionsMap;
    }

    private Dictionary<BaseState, List<Transition>>
GroupTransitionsByStates(List<Transition> transitions)
    {
        Dictionary<BaseState, List<Transition>> grouppedTransitions = new
Dictionary<BaseState, List<Transition>>();
        foreach (Transition transition in transitions)
        {
            if (!grouppedTransitions.ContainsKey(transition.Source))
                grouppedTransitions[transition.Source] = new List<Transition>();
            grouppedTransitions[transition.Source].Add(transition);
        }
        return grouppedTransitions;
    }

    private string GetSingleWord(string name, string replacer)
    {
        return Regex.Replace( name, "[^A-Za-z_0-9]", replacer );
    }

    private string GetMappedUniqueName(string name)
    {
        if (_uniqueNames == null)
            _uniqueNames = new Dictionary<string, string>();
        if (!_uniqueNames.ContainsKey(name))
            _uniqueNames[name] = GetSingleWord(name + "_" +
Guid.NewGuid().ToString(), string.Empty);
            //_uniqueNames[name] = GetSingleWord(name, string.Empty);
        return _uniqueNames[name];
    }

    private string GetEventName(string trigger)
    {
        int pos = 0;
        return GetToken(trigger, ref pos);
    }

    private string GetParametersString(string trigger, bool valuesOnly)
    {
        List< KeyValuePair<string, string> > parameters =
GetParametersPairs(trigger);
        string result = string.Empty;
        foreach (KeyValuePair<string, string> pair in parameters)
        {
            if (result.Length > 0)
                result += ", ";
            result += (valuesOnly) ? pair.Value : pair.Key + " " + pair.Value;
        }
        return result;
    }

    private List< KeyValuePair<string, string> > GetParametersPairs(string
trigger)
    {
        List<string> tokens = new List<string>();
        int pos = 0;
        string nextToken = string.Empty;
```

```
        while ((nextToken = GetToken(trigger, ref pos)) != string.Empty)
            tokens.Add(nextToken);
        if (tokens.Count > 0)
            tokens.RemoveAt(0);

        List< KeyValuePair<string, string> > result = new List<
KeyValuePair<string, string> >();
        for (int i = 0; 2 * i < tokens.Count; i++)
            result.Add( new KeyValuePair<string, string>(tokens[2 * i], tokens[2 *
i + 1]) );
        return result;
    }

    private string GetToken(string s, ref int pos)
    {
        string acceptedSymbols =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_.";
        string result = string.Empty;
        while ((pos < s.Length) && (-1 == acceptedSymbols.IndexOf(s[pos]))) pos++;
        while ((pos < s.Length) && (-1 != acceptedSymbols.IndexOf(s[pos]))) result
+= s[pos++];
        return result;
    }
    #>
```

Приложение 3. Шаблон генерации кода по диаграмме объектов

```
<#@ template
inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation"
debug="true"#>
<#@ output extension=".cs" #>
<#@ ObjectsLanguage processor="ObjectsLanguageDirectiveProcessor"
requires="fileName='%FILENAMEMARKER%'" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="System.Text.RegularExpressions" #>
namespace <#= ObjectsModel.Namespace #>
{
    public class <#= ObjectsModel.ClassName #>
    {
<#
foreach (ImTryin.ObjectsLanguage.Object obj in ObjectsModel.Objects)
{
#>
        public static <#= (obj.Namespace.Length > 0) ? obj.Namespace + "." : ""
#><#= obj.ClassName #> <#= obj.Name #> = null;
<#
}
#>

        private static void <#= GetMappedUniqueName("InitializeObjects") #>()
        {
<#
foreach (ImTryin.ObjectsLanguage.Object obj in ObjectsModel.Objects)
{
#>
            <#= obj.Name #> = new <#= (obj.Namespace.Length > 0) ? obj.Namespace +
"." : "" #><#= obj.ClassName #>(<#= obj.Parameters #>);
<#
}
#>

<#
foreach (ImTryin.ObjectsLanguage.Object obj in ObjectsModel.Objects)
{
    foreach (Composition composition in Composition.GetLinksToMembers(obj))
    {
#>
            <#= obj.Name #>.<#= composition.MemberName #> = <#=
composition.TargetObject.Name #>;
<#
    }
}
#>
        }

        private static void <#= GetMappedUniqueName("RunAutomats") #>()
        {
<#
foreach (ImTryin.ObjectsLanguage.Object obj in ObjectsModel.Objects)
{
    if (obj.HasAutomatBehavior)
    {
#>
            <#= obj.Name #>.StartStateMachine();
<#
```

```
    }
}
#>
        }

<#
if (ObjectsModel.GenerateStartPoint)
{
#>
        public static void Main(string[] args)
        {
<#
}
else
{
#>
        public static void <#= (ObjectsModel.OutputMethod.Length > 0) ?
ObjectsModel.OutputMethod : "Run" #>()
        {
<#
}
#>
            <#= GetMappedUniqueName("InitializeObjects") #>();
            <#= GetMappedUniqueName("RunAutomats") #>();
        }
    }
}

<#+
private Dictionary<String, String> _uniqueNames = null;

private string GetSingleWord(string name, string replacer)
{
    return Regex.Replace( name, "[^A-Za-z_0-9]", replacer );
}

private string GetMappedUniqueName(string name)
{
    if (_uniqueNames == null)
        _uniqueNames = new Dictionary<String, String>();
    if (!_uniqueNames.ContainsKey(name))
        _uniqueNames[name] = GetSingleWord(name + "_" + Guid.NewGuid().ToString(),
string.Empty);
    return _uniqueNames[name];
}
#>
```

Приложение 4. Обедающие философы (код сгенерирован по диаграмме классов)

```
namespace DiningPhilosophersSample
{
    public  partial class Fork
    {

        private Philosopher Owner_0c01074a1afa462bb88d382ad353f7f2 = null;

        public  Philosopher Owner
        {
            get { return Owner_0c01074a1afa462bb88d382ad353f7f2; }
            set { Owner_0c01074a1afa462bb88d382ad353f7f2 = value; }
        }

        private bool Clean_dad67b56984f4382af5e2c7351ebbbed = false;

        public  bool Clean
        {
            get { return Clean_dad67b56984f4382af5e2c7351ebbbed; }
            set { Clean_dad67b56984f4382af5e2c7351ebbbed = value; }
        }
    }
}

namespace DiningPhilosophersSample
{
    public delegate void PhilosopherEvent(Philosopher philosopher);
}

namespace DiningPhilosophersSample
{
    public delegate void PhilosopherAndForkEvent(Philosopher philosopher, Fork
fork);
}

namespace DiningPhilosophersSample
{
    public  partial class Philosopher
    {

        private
System.Collections.Generic.List<System.Collections.Generic.KeyValuePair<Philosophe
r, Fork>> forksToGive = new
System.Collections.Generic.List<System.Collections.Generic.KeyValuePair<Philosophe
r, Fork>>();

        private  const int maxSleep = 100;

        private   System.Random random = new System.Random();

        public  event PhilosopherEvent WantToEat;

        public  event PhilosopherEvent Satiated;

        private  event PhilosopherAndForkEvent ForkQueried;

        private  event EmptyEvent ForkGot;
```

```
        private Fork LeftFork_9e4d0a96c9724c8aa5869e01ec2def52 = null;

        public  Fork LeftFork
        {
            get { return LeftFork_9e4d0a96c9724c8aa5869e01ec2def52; }
            set { LeftFork_9e4d0a96c9724c8aa5869e01ec2def52 = value; }
        }

        private Fork RightFork_82aacc00ef6b425da5bb30aca00dd640 = null;

        public  Fork RightFork
        {
            get { return RightFork_82aacc00ef6b425da5bb30aca00dd640; }
            set { RightFork_82aacc00ef6b425da5bb30aca00dd640 = value; }
        }

        private  bool HasAllForks(  )
        {
            return HasFork(LeftFork) && HasFork(RightFork);
        }

        private  void QueryForks(  )
        {
            QueryFork( LeftFork );
QueryFork( RightFork );
        }

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

if ( fork.Clean )
{
    forksToGive.Add( new
System.Collections.Generic.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 != null))
    fork.Owner.ForkQueried( this, fork );

        }

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

forksToGive.Clear();
```

```
        }

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

WantToEat( this );

        }

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

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

LeftFork.Clean = false;
RightFork.Clean = false;

Satiated( this );
        }

        private  bool HasFork( Fork fork )
        {
            return this == fork.Owner;
        }
    }
}

namespace DiningPhilosophersSample
{
    public delegate void EmptyEvent();
}
```

Приложение 5. Обедающие философы (код сгенерирован по диаграмме автомата)

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

namespace DiningPhilosophersSample
{
    public partial class Philosopher
    {
        private Queue< KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906,
object[] > > eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1 = new Queue<
KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[] > >();

        private enum StateName_79421586511d411ab9f884991b9b8daf
        {
            Initial,
            Final,
            Thinking_225faac054124b73bf55f144c4756ce4,
            Starving_b9aa4f71b3344045906eb4ef0f6e2f99,
            Eating_afc6367f30984e07affa290257fbdab5
        }

        private enum EventName_b97c8de9ecc94fc284a8a677f2412906
        {
            ForkQueried_b36685609df94792b146ab8b9c25555c,
            WantToEat_796011943d25448084e3e923b3318898,
            ForkGot_df7d1c3fcf094d039de7ccd103173b2e,
            Satiated_d8e774d745034f8b8bc71322ce086256
        }

        private StateName_79421586511d411ab9f884991b9b8daf
currentState_bca2f42fbfd04e958cd8762e1013dc64 =
StateName_79421586511d411ab9f884991b9b8daf.Initial;

        public void StartStateMachine()
        {
            if (currentState_bca2f42fbfd04e958cd8762e1013dc64 !=
StateName_79421586511d411ab9f884991b9b8daf.Initial)
                throw new InvalidOperationException("State machine is already
started!");

            SubscribeEvents_5dff40e51dbb41e7bc9b5a9897afd38d();

            Initialize_1b8d3c75a3024732bff4882d3b01cfd2();

            Thread thread = new
Thread(StateMachineLoop_1c89fabe32fe434c92b07fc29a80a127);
            thread.Start();
        }

        private void Initialize_1b8d3c75a3024732bff4882d3b01cfd2()
        {
            {
                Think();
                System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is thinking...",
"StateMachineTrace" );;
```

```
SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Thinking_225faac054124b73bf55f144c4756ce4);
            }
        }

        private void Finalize_4cb5499894924025b5de650191edd41c()
        {
                UnsubscribeEvents_3d02d4b047a84bcb8b58fccbd9a898c5();
        }

        private void StateMachineLoop_1c89fabe32fe434c92b07fc29a80a127()
        {
            while (true)
            {
                KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[]
> eventToProcess;
                lock (eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1)
                {
                    while (eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1.Count == 0)
                        Monitor.Wait(eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1);
                    eventToProcess =
eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1.Dequeue();
                }
                switch (eventToProcess.Key)
                {
                    case
EventName_b97c8de9ecc94fc284a8a677f2412906.ForkQueried_b36685609df94792b146ab8b9c2
5555c :

ForkQueried_b36685609df94792b146ab8b9c25555c((Philosopher)eventToProcess.Value[0],
(Fork)eventToProcess.Value[1]);
                        break;
                    case
EventName_b97c8de9ecc94fc284a8a677f2412906.WantToEat_796011943d25448084e3e923b3318
898 :

WantToEat_796011943d25448084e3e923b3318898((Philosopher)eventToProcess.Value[0]);
                        break;
                    case
EventName_b97c8de9ecc94fc284a8a677f2412906.ForkGot_df7d1c3fcf094d039de7ccd103173b2
e :
                        ForkGot_df7d1c3fcf094d039de7ccd103173b2e();
                        break;
                    case
EventName_b97c8de9ecc94fc284a8a677f2412906.Satiated_d8e774d745034f8b8bc71322ce0862
56 :

Satiated_d8e774d745034f8b8bc71322ce086256((Philosopher)eventToProcess.Value[0]);
                        break;
                    default:
                        break;
                }
                if (currentState_bca2f42fbfd04e958cd8762e1013dc64 ==
StateName_79421586511d411ab9f884991b9b8daf.Final)
                    break;
            }
        }
```

```
        private void
SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf nextState)
        {
            if (nextState == StateName_79421586511d411ab9f884991b9b8daf.Final)
                Finalize_4cb5499894924025b5de650191edd41c();
            else
                currentState_bca2f42fbfd04e958cd8762e1013dc64 = nextState;
        }

        private void OnForkQueried_b36685609df94792b146ab8b9c25555c(Philosopher p,
Fork f)
        {
            lock (eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1)
            {
                KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[]
> pair = new KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[] >
                    (

EventName_b97c8de9ecc94fc284a8a677f2412906.ForkQueried_b36685609df94792b146ab8b9c2
5555c,
                        new object[] { p, f }
                    );
                eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1.Enqueue(pair);
                Monitor.PulseAll(eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1);
            }
        }

        private void ForkQueried_b36685609df94792b146ab8b9c25555c(Philosopher p,
Fork f)
        {
            switch (currentState_bca2f42fbfd04e958cd8762e1013dc64)
            {
                case
StateName_79421586511d411ab9f884991b9b8daf.Thinking_225faac054124b73bf55f144c4756c
e4 :
                    {
                        {
                            GiveFork(p, f);
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is thinking...",
"StateMachineTrace" );;

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Thinking_225faac054124b73bf55f144c4756ce4);
                        }
                        break;
                    }
                case
StateName_79421586511d411ab9f884991b9b8daf.Starving_b9aa4f71b3344045906eb4ef0f6e2f
99 :
                    {
                        {
                            GiveFork(p, f); QueryFork(f);
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is starving...",
"StateMachineTrace" );;

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Starving_b9aa4f71b3344045906eb4ef0f6e2f99);
                        }
```

```
                        break;
                    }
                case
StateName_79421586511d411ab9f884991b9b8daf.Eating_afc6367f30984e07affa290257fbdab5
:
                    {
                        {
                            GiveFork(p, f);
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is eating...",
"StateMachineTrace" );

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Eating_afc6367f30984e07affa290257fbdab5);
                        }
                        break;
                    }
                default:
                    break;
            }
        }
        private void OnWantToEat_796011943d25448084e3e923b3318898(Philosopher p)
        {
            lock (eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1)
            {
                KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[]
> pair = new KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[] >
                    (

EventName_b97c8de9ecc94fc284a8a677f2412906.WantToEat_796011943d25448084e3e923b3318
898,
                        new object[] { p }
                    );
                eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1.Enqueue(pair);
                Monitor.PulseAll(eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1);
            }
        }

        private void WantToEat_796011943d25448084e3e923b3318898(Philosopher p)
        {
            switch (currentState_bca2f42fbfd04e958cd8762e1013dc64)
            {
                case
StateName_79421586511d411ab9f884991b9b8daf.Thinking_225faac054124b73bf55f144c4756c
e4 :
                    {
                        if (!HasAllForks())
                        {
                            QueryForks();
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is starving...",
"StateMachineTrace" );;

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Starving_b9aa4f71b3344045906eb4ef0f6e2f99);
                        }
                        else if (HasAllForks())
                        {
                            Eat();
```

```
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is eating...",
"StateMachineTrace" );

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Eating_afc6367f30984e07affa290257fbdab5);
                        }
                        break;
                    }
                default:
                    break;
            }
        }
        private void OnForkGot_df7d1c3fcf094d039de7ccd103173b2e()
        {
            lock (eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1)
            {
                KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[]
> pair = new KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[] >
                    (

EventName_b97c8de9ecc94fc284a8a677f2412906.ForkGot_df7d1c3fcf094d039de7ccd103173b2
e,
                        new object[] {  }
                    );
                eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1.Enqueue(pair);
                Monitor.PulseAll(eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1);
            }
        }

        private void ForkGot_df7d1c3fcf094d039de7ccd103173b2e()
        {
            switch (currentState_bca2f42fbfd04e958cd8762e1013dc64)
            {
                case
StateName_79421586511d411ab9f884991b9b8daf.Starving_b9aa4f71b3344045906eb4ef0f6e2f
99 :
                    {
                        if (!HasAllForks())
                        {
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is starving...",
"StateMachineTrace" );;

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Starving_b9aa4f71b3344045906eb4ef0f6e2f99);
                        }
                        else if (HasAllForks())
                        {
                            Eat();
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is eating...",
"StateMachineTrace" );

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Eating_afc6367f30984e07affa290257fbdab5);
                        }
                        break;
                    }
                default:
                    break;
```

```
            }
        }
        private void OnSatiated_d8e774d745034f8b8bc71322ce086256(Philosopher p)
        {
            lock (eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1)
            {
                KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[]
> pair = new KeyValuePair< EventName_b97c8de9ecc94fc284a8a677f2412906, object[] >
                    (

EventName_b97c8de9ecc94fc284a8a677f2412906.Satiated_d8e774d745034f8b8bc71322ce0862
56,
                        new object[] { p }
                    );
                eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1.Enqueue(pair);
                Monitor.PulseAll(eventQueue_c3c3d0bba4ee46be8ea5647cf54fd7d1);
            }
        }

        private void Satiated_d8e774d745034f8b8bc71322ce086256(Philosopher p)
        {
            switch (currentState_bca2f42fbfd04e958cd8762e1013dc64)
            {
                case
StateName_79421586511d411ab9f884991b9b8daf.Eating_afc6367f30984e07affa290257fbdab5
:
                    {
                        {
                            GiveQueriedForks(); Think();
                            System.Diagnostics.Trace.WriteLine( "Philosopher [" +
Thread.CurrentThread.ManagedThreadId.ToString() + "] is thinking...",
"StateMachineTrace" );;

SetNewState_381f82544e1d4b7a8c316fbb2f7fc7fd(StateName_79421586511d411ab9f884991b9
b8daf.Thinking_225faac054124b73bf55f144c4756ce4);
                        }
                        break;
                    }
                default:
                    break;
            }
        }

        private void SubscribeEvents_5dff40e51dbb41e7bc9b5a9897afd38d()
        {
            ForkQueried += OnForkQueried_b36685609df94792b146ab8b9c25555c;
            WantToEat += OnWantToEat_796011943d25448084e3e923b3318898;
            ForkGot += OnForkGot_df7d1c3fcf094d039de7ccd103173b2e;
            Satiated += OnSatiated_d8e774d745034f8b8bc71322ce086256;
        }

        private void UnsubscribeEvents_3d02d4b047a84bcb8b58fccbd9a898c5()
        {
            ForkQueried -= OnForkQueried_b36685609df94792b146ab8b9c25555c;
            WantToEat -= OnWantToEat_796011943d25448084e3e923b3318898;
            ForkGot -= OnForkGot_df7d1c3fcf094d039de7ccd103173b2e;
            Satiated -= OnSatiated_d8e774d745034f8b8bc71322ce086256;
        }
    }
}
```

Приложение 6. Обедающие философы (код сгенерирован по диаграмме объектов)

```
namespace DiningPhilosophersSample {
    public class DiningPhilosophers {
        public static DiningPhilosophersSample.Fork fork1 = null;
        public static DiningPhilosophersSample.Philosopher philosopher1 = null;
        public static DiningPhilosophersSample.Philosopher philosopher2 = null;
        public static DiningPhilosophersSample.Philosopher philosopher3 = null;
        public static DiningPhilosophersSample.Philosopher philosopher4 = null;
        public static DiningPhilosophersSample.Fork fork2 = null;
        public static DiningPhilosophersSample.Fork fork4 = null;
        public static DiningPhilosophersSample.Fork fork5 = null;
        public static DiningPhilosophersSample.Fork fork3 = null;
        public static DiningPhilosophersSample.Philosopher philosopher5 = null;
        private static void InitializeObjects_2fdd4dd3d2e048a98c54913b2db9cd77()
        {
            fork1 = new DiningPhilosophersSample.Fork();
            philosopher1 = new DiningPhilosophersSample.Philosopher();
            philosopher2 = new DiningPhilosophersSample.Philosopher();
            philosopher3 = new DiningPhilosophersSample.Philosopher();
            philosopher4 = new DiningPhilosophersSample.Philosopher();
            fork2 = new DiningPhilosophersSample.Fork();
            fork4 = new DiningPhilosophersSample.Fork();
            fork5 = new DiningPhilosophersSample.Fork();
            fork3 = new DiningPhilosophersSample.Fork();
            philosopher5 = new DiningPhilosophersSample.Philosopher();
            fork1.Owner = philosopher2;
            philosopher1.RightFork = fork1;
            philosopher1.LeftFork = fork5;
            philosopher2.LeftFork = fork1;
            philosopher2.RightFork = fork2;
            philosopher3.LeftFork = fork2;
            philosopher3.RightFork = fork3;
            philosopher4.LeftFork = fork3;
            philosopher4.RightFork = fork4;
            fork2.Owner = philosopher3;
            fork4.Owner = philosopher5;
            fork5.Owner = philosopher1;
            fork3.Owner = philosopher4;
            philosopher5.LeftFork = fork4;
            philosopher5.RightFork = fork5;
        }
        private static void RunAutomats_598ef02735d641fb82538977bc771c97()
        {
            philosopher1.StartStateMachine();
            philosopher2.StartStateMachine();
            philosopher3.StartStateMachine();
            philosopher4.StartStateMachine();
            philosopher5.StartStateMachine();
        }
        public static void Main(string[] args)
        {
            InitializeObjects_2fdd4dd3d2e048a98c54913b2db9cd77();
            RunAutomats_598ef02735d641fb82538977bc771c97();
        }
    }
}
```