Столяров Леонид Владимирович

ТРАНСЛЯЦИЯ ОПИСАНИЙ АВТОМАТОВ, ПРЕДСТАВЛЕННЫХ В ФОРМАТЕ *MICROSOFT VISIO*, В ИСХОДНЫЙ КОД НА ЯЗЫКЕ *C*

Аннотация

Цель работы – создание инструментального средства для трансляции описаний графов переходов автоматов, представленных в формате *MS Visio,* в программный код на языке *C*. В качестве языка реализации указанного средства выбран язык *C#*, предназначенный для платформы *.NET*, так как он обеспечивает хорошую поддержку COM-технологии, используемой при взаимодействии с *MS Visio*.

Программа транслятора может быть запущена в двух режимах. В первом из них трансляция производится интерактивно, и вся информация считывается из командной строки. Этот режим удобен при использовании в средствах автоматической сборки (например *make*, менеджер проектов *MS Visual Studio*). Во втором режиме интерфейсом программы является диалоговое окно.

Ключевые слова: автоматное программирование, графы переходов, инструментальное средство.

ВВЕДЕНИЕ

Существует подход к созданию программ с использованием конечных автоматов, названный автоматным программированием [1–3]. С его помощью удобно реализуются различные системы логического и событийного управления [4].

При использовании этого подхода разрабатываются графы переходов автоматов с последующей их трансляцией в исходный код программы. Важное достоинство автоматного программирования состоит в возможности разделения этапов проектирования и реализации программы, в том числе и для того, чтобы эти этапы могли выполняться различными людьми. Граф переходов является графическим отражением алгоритма в терминах состояний и переходов. Поэтому при разработке программ на основе рассматриваемого подхода графы переходов документируют алгоритмы, что чрезвычайно важно для дальнейшего сопровождения программ. Достоинство автоматного подхода состоит также и в том, что программы, построенные на его основе, в отличие от программ, написанных традиционным путем, могут быть достаточно просто формально верифицированы на основе метода *Model Checking* [5].

Визуализация поведения программы позволяет ускорить ее разработку и отладку, а также отделить алгоритмическую (логическую, управляющую) часть от остального кода. В автоматном программировании, как и в теории автоматов, используются три понятия: *входное воздействие, состояние* и *выходное воздействие.* При этом *состояния выделяются* на этапе проектирования в *явном виде*.

Для разработки графов переходов могут использоваться распространенные средства моделирования, в частности, среда *Microsoft Visio*. Этот редактор позволяет строить различные схемы, использующие элементы шаблонов, как встроенных, так и загружаемых из внешних файлов.

При создании проектов с использованием конечных автоматов возникает проблема трансляции графов переходов в исходный код программы на конкретном языке программирования.

Существуют методы как ручной трансляции (построение автоматной программы по графу переходов), так и средства автоматической генерации кода. Применение ручных методов может приводить к ошибкам. Поэтому автоматические методы предпочтительнее. При этом существенную роль играет организация взаимодействия средства трансляции с автором программы (настройка режимов трансляции, способ запуска, сообщения об ошибках трансляции и т. д.).

Известен ряд инструментальных средств поддержки автоматного программирования: *Visio2Switch* [6] – для языка *C*, *MetaAuto* [7] – для множества языков, задаваемых шаблонами, *UniMod* [8] – для языка *Java*.

Эти средства имеют как достоинства, так и недостатки. Например, у средства *MetaAuto* недостаточно развита диагностика ошибок в графе переходов, а также низка надежность, так как сгенерированный код не всегда компилируется. Для средства *Visio2Switch* невозможна или крайне сложна интеграция в автоматизированные процессы сборки приложений.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью настоящей работы является создание инструментального средства для трансляции описаний графов переходов, представленных в формате *MS Visio* в исходный код на языке *C* и другие формы представления данных, например язык *XML*.

При этом должны быть учтены требования, предъявляемые к средствам, работающим с автоматами:

1. Автоматическое документирование поведения программ.
2. Повышение эффективности отладки и тестирования программ.
3. Повышение эффективности ручной и автоматической генерации кода.

Также должно быть обеспечено:

1. Качество и надежность генерируемого кода.
2. Диагностика ошибок в исходных данных.
3. Интеграция в автоматические процессы сборки приложений.

Для реализации разрабатываемого средства выбран язык C#, предназначенный для платформы *.NET*. Он предоставляет хорошие возможности для работы с *COM*-технологией, необходимой для взаимодействия модулей транслятора с *MS Visio*. Ее поддержка на платформе *.NET* сильно облегчает этот процесс.

Граф переходов в редакторе *MS Visio* состоит из описания автомата, его элементов, состояний и переходов между состояниями. Этих элементов нет в стандартной библиотеке *MS Visio*. Поэтому одной из задач работы является разработка файла с шаблонами, необходимыми для построения графов переходов автоматов.

ФОРМАТ ГРАФОВ ПЕРЕХОДОВ АВТОМАТОВ

Рассмотрим формат описания графов переходов автомата, который соблюдается при их построении и обрабатывается транслятором. Граф переходов автомата [2, 4] состоит из нескольких основных элементов: состояния, групповые состояния, описание входных и выходных воздействий, переходы (рис. 1).

Рассмотрим основные элементы, используемые при построении графа переходов автомата.

Состояние имеет имя, список других автоматов для запуска из этого состояния и список выходных воздействий, формируемых в этом состоянии.

Событие – один из видов входных воздействий, которые обеспечивают вызов автомата. Оно обычно используется в условиях перехода.

Групповые состояния (группа состояний) предназначены для создания групповых переходов и могут содержать в себе состояния или другие группы состояний.

Переход соединяет два состояния или две группы состояний. Переход помечается условием, представленным в виде произвольного логического выражения, в котором могут использоваться входные переменные, и списком выходных воздействий, которые выполняются при переходе.

Описание входных или выходных воздействий – это элемент, комментирующий какое-либо входное или выходное воздействие. Имена функций для входных или выходных воздействий в традиционно написанных программах часто не являются интуитивно понятными. Поэтому человек, разбирающийся в описании графа переходов автомата или в сгенерированном по нему исходном коде, не может быстро и легко разобраться в том, какую функциональность реализуют те или иные функции воздействий. Это затрудняет процесс отладки и совместной разработки программы разными людьми. Поэтому данный элемент является важным для понимания графов переходов. В таблице описаний есть два поля: имя входного (или выходного) воздействия и словесное описание, соответствующее воздействию. Это словесное описание будет использоваться как разработчиком при разборе графа переходов, так и для составления комментариев в сгенерированном коде.

Важным является также элемент *«Описание автомата»*. Он содержит имя автомата и его словесное описание. Имя автомата – это строка, состоящая из английских символов и цифр, которая будет применяться для идентификации автомата после трансляции. Описание автомата – это произвольная строка, поясняющая назначение данного автомата.

В целях устранения путаницы, транслятор считает, что на каждой странице *Visio* определен только один автомат. Страница *Visio* должна называться так же, как и автомат, определенный на ней. Поскольку автомат не имеет смысла без имени и хотя бы краткого описания, этот элемент должен обязательно присутствовать в исход-

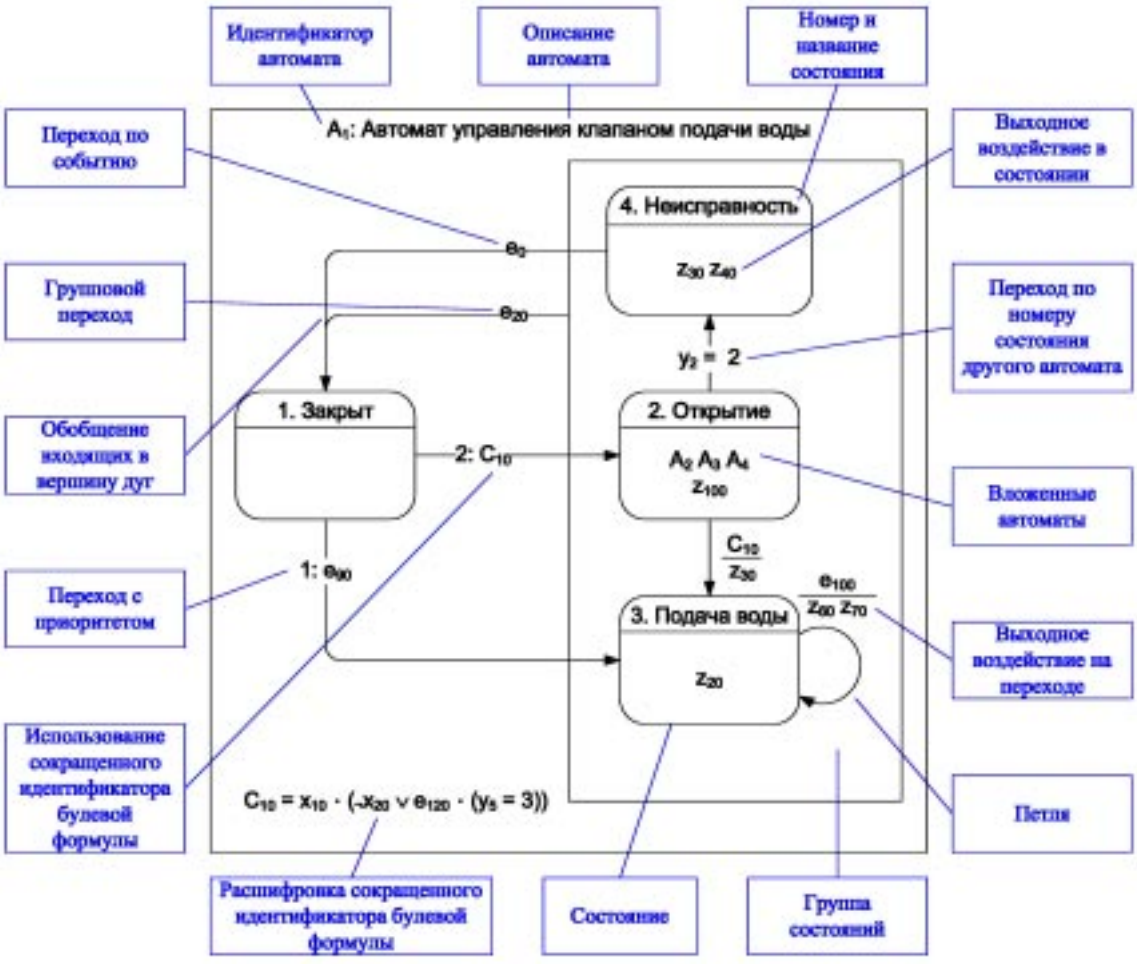

Рис. 1. Нотация графов переходов автомата

Рис. 2. Алгоритм процесса трансляции

ном документе *Visio*, иначе транслятор не будет работать с этим документом и выдаст ошибку.

ПРОЦЕСС ТРАНСЛЯЦИИ

Рассмотрим алгоритм процесса трансляции графов переходов. Автоматная схема (один или несколько графов переходов), содержащаяся в исходном документе *Visio,* преобразуется в исходный код для дальнейшего использования в автоматном проекте, а также сохраняется в *XML*-формате.

Этот процесс можно разделить на четыре этапа (рис. 2):

1. Чтение данных из исходного документа *Visio*.

2. Преобразование данных из документа *Visio* в модель, основанную на состояниях и переходах (здесь и далее – внутренняя модель представления автомата, или просто внутренняя модель). Эта модель применяется внутри транслятора для представления автоматной схемы с помощью классов языка C#.

3. Генерация файла *XML*-формата на основе внутренней модели.

4. Генерация файлов с исходным кодом на языке *C* на основе внутренней модели.

ВНУТРЕННЯЯ МОДЕЛЬ ПРЕДСТАВЛЕНИЯ АВТОМАТА

Как отмечалось выше, в трансляторе используется внутренняя модель представления автомата, основанная на классах языка C#. Для представления элементов этой модели в программе определены классы для описания автоматного проекта, автомата, состояния, перехода и других элементов автоматной схемы (рис. 3). Детальная структура и взаимодействие основных классов приведены на рис. 4.

В классе *представления автомата (Automaton)* содержится имя автомата, его описание, набор используемых состояний и набор описаний входных и выходных воздействий этого автомата.

Класс представления автоматного проекта (AutomataProject) содержит множество автоматов, входящих в этот проект. В этот проект входят все автоматы, определенные в исходном документе *Visio*.

В *классе представления состояний (State)* содержится множество переходов, соединяющих данное состояние с другими состояниями того же автомата, список вложенных автоматов, имя этого состояния, список выходных воздействий в этом состоянии.

В *классе представления переходов (Transition)* содержатся состояния, которые соединяет переход, список выходных воздействий, совершаемых при переходе, условие перехода.

В *классе представления входных и выходных воздействий (InputOutputDefinition)* содержится имя воздействия и описание воздействия, поясняющее имя.

Рис. 3. Общая структура внутренней модели представления автомата

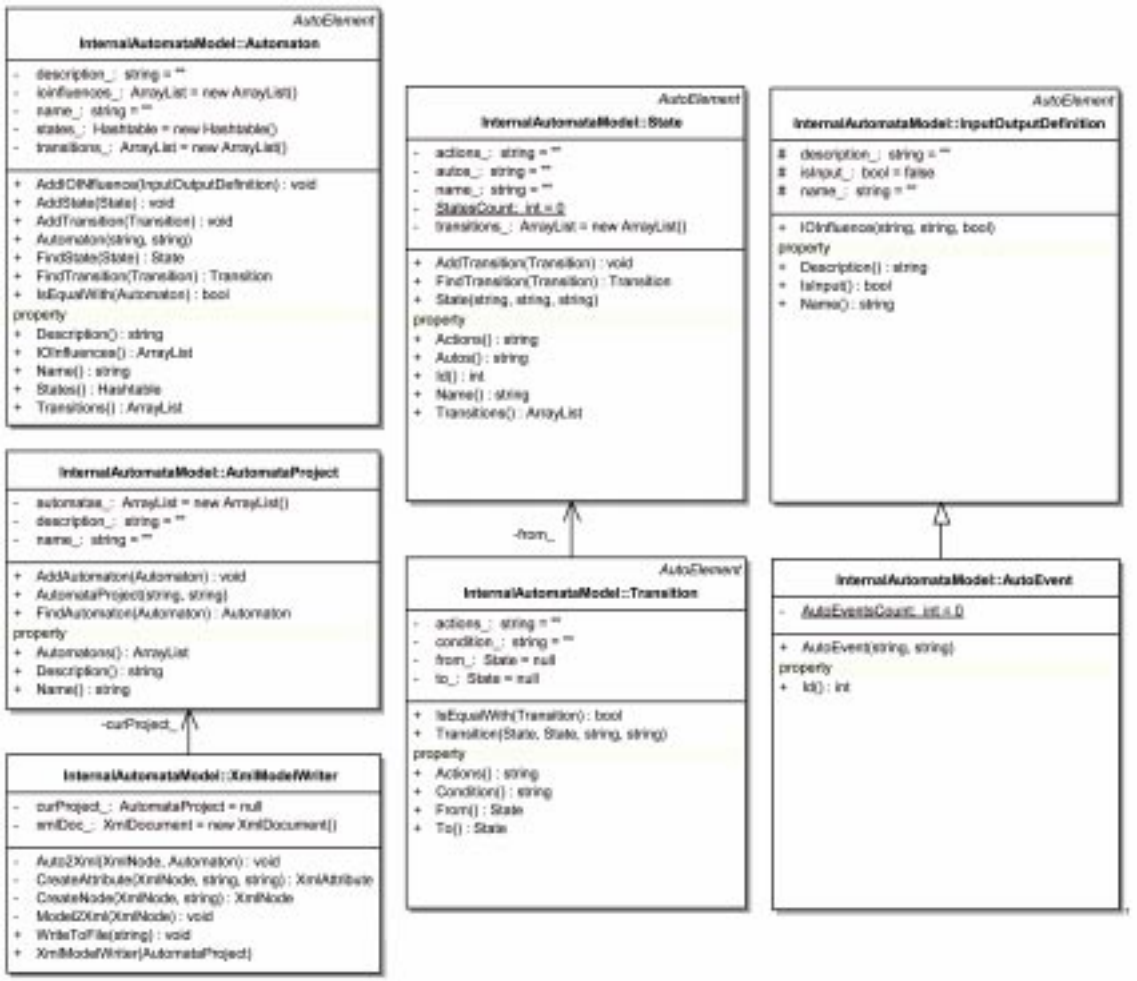

Рис. 4. UML-диаграмма основных классов внутренней модели представления автомата

Класс представления события (AutoEvent) описывает один из видов входных воздействий – событие. Так как способ его трансляции сильно отличается от способа трансляции входных переменных (другого типа входных воздействий), для его описания создан отдельный класс.

Класс XmlModelWriter предоставляет функции для генерации *XML*-файла на основе автоматной схемы.

Поскольку каждый элемент графа переходов с точки зрения *Visio* является фигурой, полученной из шаблонного файла, в каждом классе, представляющем элемент внутренней модели представления автомата, предусмотрен конструктор, принимающий в качестве параметра элемент из шаблонных фигур *Visio*.

ЧТЕНИЕ ДАННЫХ ИЗ ИСХОДНОГО ДОКУМЕНТА *MS VISIO*

Первым этапом процесса трансляции является чтение данных из исходного документа *Visio*. Для реализации этого этапа был применен подход, основанный на непосредственном взаимодействии с редактором *MS Visio*.

Такое взаимодействие выполняется в трансляторе при помощи *COM*-технологии. Вместе с установкой на компьютер *Microsoft Office Visio* устанавливается также и соответствующий *COM*-объект. Интерфейсы *COM*-объекта *MS Visio* позволяют открывать любые файлы, сохраненные в формате редактора, и получать список всех элементов схемы в открытом файле.

Рис. 5. Взаимодействие с *MS Visio*

Взаимодействие с редактором *Visio* в программе происходит следующим образом (рис. 5):

1. Создание экземпляра (*COM*-объекта) *Visio*.
2. Открытие в экземпляре исходного документа.
3. Получение данных из исходного документа.
4. Завершение работы *COM*-объекта *Visio,* так как связь с ним в дальнейшем больше не требуется.

Внутренняя структура исходного документа *Visio* – это иерархический набор фигур. Каждая фигура имеет такие параметры, как имя, тип, фигура-родитель, текст на фигуре и другие.

Согласно структуре исходного документа *Visio,* все элементы графа переходов автомата являются фигурами. Эти фигуры загружаются из специального файла шаблонных фигур для построения автоматных схем, входящего в комплект транслятора.

ПРЕОБРАЗОВАНИЕ ИСХОДНОГО ДОКУМЕНТА *VISIO* ВО ВНУТРЕННЮЮ МОДЕЛЬ ПРЕДСТАВЛЕНИЯ АВТОМАТА

После установления связи с *COM*-объектом *Visio* начинается процесс импорта данных и создания объектов представления автоматов (рис. 5, 6). При этом считается, что если на некоторой странице исходного документа есть элемент «описание автомата», то следует рассматривать остальное содержание страницы как его определение. В результате создается объект представления автомата, в который записываются его имя и описание.

После этого список фигур *Visio* на данной странице перебирается в цикле. Для каждой фигуры определяется ее тип. Он может быть определен как состояние, соединитель, расширенный соединитель, группа состояний или описание входных или выходных воздействий.

Рис. 6. Генерация внутренней модели представления автомата

Если фигура – состояние, то по ней генерируется объект состояния с соответствующими именем, описанием и другими данными. Затем состояние добавляется в список состояний текущего автомата (рис. 6).

Если фигура – описание входного или выходного воздействия, то из нее создается объект описания, которое затем добавляется в список описаний воздействий текущего автомата. Оно будет использовано в дальнейшем

при генерации кода для автоматического комментирования кода.

Если фигура – соединение или расширенное соединение, то создается объект перехода. Из данных *Visio* можно узнать исходную фигуру соединителя и к какой фигуре он присоединяется. После определения этих двух фигур из них создаются объекты состояний, которые затем указываются в объекте перехода.

После того как все данные переходов и состояний сохранены в текущем объекте автомата, в список переходов каждого состояния данного автомата добавляются те переходы, которые исходят из этого состояния.

На этом преобразование одного графа переходов автомата в формате *Visio* в модель, основанную на переходах и состояниях, заканчивается.

Аналогичный процесс повторяется для всех других автоматов в исходном документе. Множество преобразованных в модель автоматов хранится в объекте *AutomataProject*, который также имеет имя. Таким образом, после окончательного преобразования всех автоматов в модель получается объект автоматного проекта с массивом всех его автоматов и именем. После этого исходные данные из документа *Visio* больше не потребуются в процессе трансляции.

Как следует из изложенного, в данной реализации алгоритма процесс импорта данных из *Visio*, занимающий наибольшее время из всех этапов трансляции, является однопроходным. Это ускоряет преобразование, по сравнению с инструментальным средством *MetaAuto*, в котором процесс импорта данных является двухпроходным.

ГЕНЕРАЦИЯ ФАЙЛА *XML*-ФОРМАТА НА ОСНОВЕ ВНУТРЕННЕЙ МОДЕЛИ

Следующим этапом процесса трансляции является генерация файла в *XML*-формате на основе внутренней модели. В этом файле описываются все автоматы транслируемого документа. В каждом автомате, в свою очередь, описывается каждое состояние, содержится список состояний, с которыми это состояние связано, условия перехода и список выходных воздействий. Наличие таких файлов достаточно для описания автоматной схемы. Поэтому они могут использоваться в качестве исходных данных для работы других инструментальных средств или для хранения автоматной схемы в *XML*-формате.

ГЕНЕРАЦИЯ ФАЙЛОВ С ИСХОДНЫМ КОДОМ НА ЯЗЫКЕ *C* НА ОСНОВЕ ВНУТРЕННЕЙ МОДЕЛИ

Следующим этапом процесса трансляции после генерации *XML*-файла является генерация исходного программного кода на языке *C*, представленного в виде набора файлов (рис. 7).

Для удобства пользователя эти файлы размещаются в папке, которая называется так же, как и исходный документ *Visio*.

Процесс кодогенерации происходит на основе файлов – шаблонов, специфичных для каждого языка и размещаемых в специальной директории, выбираемой пользователем. Эти файлы могут иметь произвольное содержание, но должны содержать специальные метки (например, $automata_functions). В процессе трансляции эти метки будут заменены на соответствующие участки сгенерированного кода. Такой подход позволяет любому пользователю настроить стиль генери-

Рис. 7. Генерация файлов с исходным кодом на языке *C*

руемых файлов так, как ему больше нравится, не прикладывая особых усилий.

Кодогенератор устроен таким образом, что генерацию частей исходного кода (таких как функция, условный оператор, оператор присваивания, вызов функции и т. д.) он производит не самостоятельно, а обращаясь к объекту генерации кода, который тоже передается генератору в качестве входных данных. Таким объектом может быть объект любого класса языка C#, поддерживающего интерфейс генерации кода. Такой подход позволяет генерировать исходный код на разных языках программирования, имея объект генерации кода для каждого из этих языков. Предполагается, что генерация автоматных программ с использованием разных языков программирования будет реализована в дальнейшем.

Генерация файлов с исходным кодом является последней стадией работы программы. Поэтому после данного этапа модуль программы, отвечающий за трансляцию, завершает свою работу.

ИНТЕРФЕЙС ПРОГРАММЫ

При выполнении работы особое внимание было уделено пользовательскому интерфейсу транслятора. Программа может быть запущена в двух режимах, выбираемых ключом командной строки. В первом из них трансляция производится неинтерактивно, и вся информация считывается из командной строки (рис. 8). Этот режим удобен при использовании в средствах автоматической сборки (например, *make*, менеджер проектов *MS Visual Studio*).

Второй режим задается ключом командной строки –gui. В нем интерфейсом программы является диалоговое окно, в котором пользователь может задавать параметры, пользуясь элементами графического интерфейса *.NET*. Если же программа запущена без параметров командной строки, то она открывается в режиме графического интерфейса.

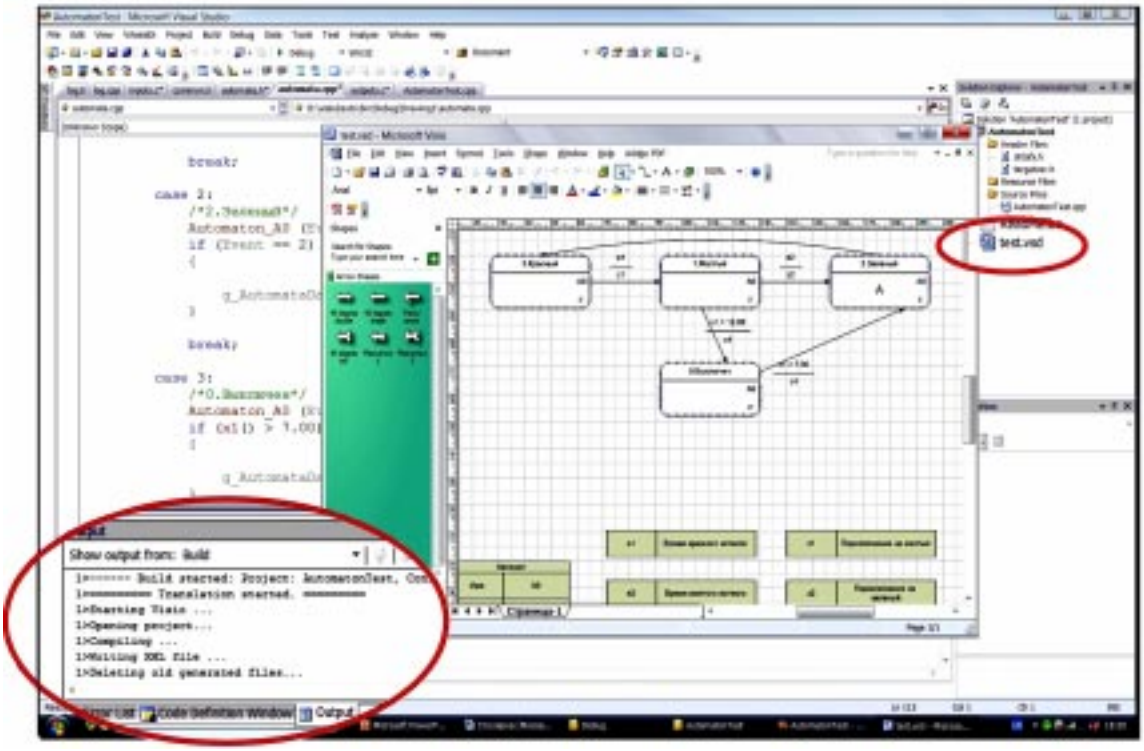

Рис. 8. Пример работы транслятора в режиме интеграции с *MS Visual Studio*

АРХИТЕКТУРА ТРАНСЛЯТОРА

Классы ядра транслятора

Интерфейс генерации кода ICodeWriter реализуют те классы, объекты которых будут использоваться при генерации исходного кода на конкретном языке программирования, например класс *C_CodeWriter*.

Класс C_CodeWriter содержит функции генерации частей кода на языке программирования *C*. Поскольку он поддерживает интерфейс генерации кода (*ICodeWriter*), объект этого класса передается в класс *CodeGenerator*.

Класс CodeGenerator является вспомогательным для процесса генерации набора файлов с исходным кодом. Он генерирует конкретные части исходного кода на конкретном языке (в данном случае *C*), которые впоследствии будут использованы при генерации набора выходных файлов с кодом.

Класс OutputCodeWriter предназначен для генерации набора файлов с исходным кодом на основе файлов – шаблонов.

Класс Compiler содержит функции преобразования автоматной схемы *Visio* во внутреннюю модель представления автомата.

Класс ErrorSystem предназначен для работы системы обработки ошибок, найденных в автоматной схеме. В частности, в нем содержится множество всех сообщений о найденных ошибках.

Классы интерфейса пользователя

Класс ConsoleInstance обеспечивает работу программы в режиме командной строки.

Класс GUIInstance обеспечивает работу программы в режиме графического интерфейса пользователя.

Вспомогательные классы

Класс LoggingDebugSystem обеспечивает работу системы отладки в трансляторе. Он осуществляет управление лог-файлами программы.

СИСТЕМА ОБРАБОТКИ ДЕФЕКТОВ АВТОМАТНОЙ СХЕМЫ

При разработке транслятора особое внимание было уделено системе обработки дефектов проектирования автоматной схемы, совершенных пользователем и найденных транслятором в графе переходов. Эти дефекты подразделяются на два типа: ошибка и предупреждение. Ошибка – это некий найденный дефект схемы, без исправления которого разработчиком нельзя продолжить формальную трансляцию. Предупреждение – это дефект, с которым продолжить формальную трансляцию можно. Предупреждение, скорее всего, является следствием высокоуровневой логической ошибки в исходной схеме (поэтому в данном описании подчеркнута формальность трансляции). В процессе трансляции все ошибки и предупреждения выводятся в консоль или главное окно транслятора.

В режиме интеграции со средствами сборки приложений при возникновении ошибок, транслятор формирует единицу в качестве возвращаемого значения программы. Это позволяет вызвавшему средству сборки приложений распознать, что в трансляции произошла ошибка и остановить процесс сборки. Формат вывода ошибки или предупреждения идентичен формату компилятора *Visual Studio*. Формат содержит имя исходного документа, имя автомата, ошибочный элемент схемы, номер (уникальный идентификатор) ошибки и строку с сообщением об ошибке.

Ниже перечислены основные типы определяемых ошибок и предупреждений.

1. Стрелка перехода соединяет не состояния, а другие фигуры *Visio*.
2. Стрелка перехода не присоединена к какой-либо фигуре.
3. В автомате найдены совпадающие элементы.
4. У автоматного состояния отсутствует имя.
5. У перехода от одного состояния к другому нет условия выполнения.

РЕЗУЛЬТАТЫ РАБОТЫ И ПЛАНЫ

В результате работы создано инструментальное средство для трансляции графов переходов автоматов, представленных в формате *Microsoft Visio*, в исходный код на языке *C*.

Средство обеспечивает сохранение исходных данных в *XML*-файл, диагностику ошибок в исходных данных, интеграцию в автоматические процессы сборки приложений (например, *make*-файлы, *MS Visual Studio build system*).

Планируется расширить функциональные возможности средства, например в части трансляции в другие языки программирования, что предполагается сделать с помощью подключаемых модулей (*plug in*). Это позволит любому желающему, владеющему программированием под *.NET*, написать модуль для любого известного ему языка программирования, и, таким образом, реализовать трансляцию графов переходов автоматов в код на выбранном языке, не используя исходный код транслятора.

РАЗМЕЩЕНИЕ В СЕТИ ИНТЕРНЕТ

Сайт программы с документацией, оперативной информацией, обновлениями, исходными текстами и тестовыми файлами расположен на сервере программного обеспечения с открытым исходным кодом *SourceForge* по адресу http://txlib.wiki.sourceforge.net/Visio2Auto. Информация также представлена на сайте кафедры «Технологии программирования» СПбГУ ИТМО по адресу http://is.ifmo.ru/automata_school. Для оперативной работы с сообщениями об ошибках в программе и предложениями по ее доработке (feature requests) используются трекеры сайта *SourceForge*.

Литература

1. *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

2. *Поликарпова Н.И., Шалыто А.А.* Автоматное программирование. СПб.: Питер, 2009.

3. *Шалыто А.А., Наумов Л.А.* Реализация автоматов в объектно-ориентированных программах // Искусственный интеллект, 2004. № 4. http://is.ifmo.ru/works/_aut_oop.pdf

4. *Шалыто А.А., Туккель Н.И.* Реализация автоматов при программировании событийных систем // Программист, 2004. № 2. http://is.ifmo.ru/works/evsys/

5. *Егоров К.В., Шалыто А.А.* Методика верификации автоматных программ // Информационно-управляющие системы, 2008. № 5. http://is.ifmo.ru/works/_egorov.pdf

6. *Головешин А.* Инструментальное средство *Visio2Switch.* http://is.ifmo.ru/progeny/visio2switch

7. *Канжелев С.Ю., Шалыто А.А.* Преобразование графов переходов, представленных в формате *MS Visio* в исходные коды программ для различных языков программирования (инструментальное средство *MetaAuto*). Проектная документация. http://is.ifmo.ru/projects/metaauto

8. *Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А.* UML. Switch-технология. Eclipse // Информационно-управляющие системы, 2004. № 6. http://is.ifmo.ru/works/uml-switch-eclipse/

Abstract

The open-source tool for translation of discrete finite automata descriptions from *Microsoft Visio* format to source code in *C* language is developed. There is a number of similar tools, but some of them have poor error diagnostics while translation, some have poor stability and quality of code they generate, and some of them aren't open-source. The aim of this project is the development of tool without these drawbacks. The tool itself is written on *C#* language for *.NET* platform.

Столяров Леонид Владимирович, ученик 8 класса, лицей «Вторая школа»,

lnd1212@rambler.ru