Статья опубликована в материалах конференции

“Software Engineering Conference (Russia) – 2006” (SEC (R) 2006), с. 60 – 63.

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ КОДА ПРОГРАММ С ЯВНЫМ ВЫДЕЛЕНИЕМ СОСТОЯНИЙ

AUTOMATIC CODE GENERATION OF THE PROGRAMS WITH THE OBVIOUS STATE EXTRACTING

Канжелев Сергей Юрьевич
магистрант СПбГУ ИТМО
email: kanzhser@rain.ifmo.ru

Шалыто Анатолий Абрамович
доктор технических наук
профессор СПбГУ ИТМО
email: shalyto@mail.ifmo.ru

Аннотация

В последнее время весьма актуален вопрос о визуальном конструировании программ. При этом важным является проблема автоматической генерации кода по графическим моделям. В работе рассматривается подход к автоматической генерации кода программ с явным выделением состояний на любом априори заданном языке программирования по графам переходов автоматов программ.

***Ключевые слова**: автоматное программирование; программирование с явным выделением состояний; автоматизация; граф переходов.*

1. Введение

В последнее время весьма актуален вопрос о визуальном конструировании программ [1]. При этом важным является проблема автоматической генерации кода по графическим моделям. Модели можно разделить на два класса: статические и динамические.

Среди статических моделей объектно-ориентированных программ основной является диаграмма классов. Многие инструментальные средства генерируют «скелет» кода по диаграммам классов.

Динамические свойства рассматриваемого класса программ наиболее эффективно описывают диаграммы состояний (графы переходов).

В любой программе имеются состояния. Однако программы по структуре могут быть разбиты на два класса: с явно выделенными состояниями и без них. В первом случае для описания поведения программ можно использовать аппарат теории автоматов и диаграммы состояний *UML* или им аналогичные. Во втором случае программы содержат большое число флагов, и изменения в одних частях программы часто приводят к непредвиденным изменениям в других ее частях, так как состояния распределены по программе и явно не заданы.

В соответствии с парадигмой автоматного программирования [2], программы предлагается строить так, как выполняется автоматизация технологических (и не только) процессов. При этом программа строится из следующих компонентов: источники входных воздействий (события и входные переменные), система взаимодействующих автоматов, объекты управления и обратные связи от объектов к автоматам.

Известны различные инструментальные средства, генерирующие код по графам переходов, которые описаны, например, в работах [3–5]. Указанные средства являются специализированными – строят код для одного – двух языков программирования.

Однако на практике используются различные аппаратные решения, в которых применяются различные языки программирования. Так, например, для одних систем, часто используются процедурные языки типа ассемблера и *C*, а для других систем – объектно-ориентированные языки *C++*, *Java* и *C#*.

Цель настоящей работы состоит в описании подхода к генерации кода программ с явным выделением состояний на любом априори заданном языке программирования.

На основе этого подхода было создано инструментальное средство *MetaAuto* [6], которое позволяет преобразовывать изображения графов переходов, представленных с помощью редактора *MS Visio*, в исходные коды программ на любых языках программирования, для которых предварительно созданы соответствующие шаблоны. В ходе работы над этим инструментальным средством в качестве примера были созданы шаблоны для трех языков программирования (*C*, *C#* и *Turbo Assembler*).

2. Этапы генерации кода

2.1. Декомпозиция задачи

Как отмечалось выше, описываемый подход предполагает построение исходного кода на различных языках программирования. Для реализации этого требования разумно разделить этап преобразования графической модели в исходный код на два: преобразование графической модели в некий общий формат, например, XML-описание графа переходов, и преобразование из этого формата в различные языки программирования.

Такое разделение приводит к большей платформенной независимости. При этом легко перейти от одного редактора графов переходов на другой, реализовав для нового редактора процедуру преобразования в XML-формат.

Для облегчения работы с данными, содержащимися в графе переходов, имеет смысл использовать программное представление общего формата. Под программным представлением понимается библиотека классов, изоморфная по своей семантике данным общего формата. Библиотека предназначена для упрощения процесса чтения, изменения и сохранения графов переходов автомата различными программами: валидаторами, генераторами кода, другими программами пользователя.

Таким образом, этап генерации кода может быть разбит на три: преобразование графа переходов в программное представление, изоморфное преобразование программного представления в общий формат и, наконец, преобразование из общего формата в исходный код программы. На рис. 1 изображены этапы генерации исходного кода программы.

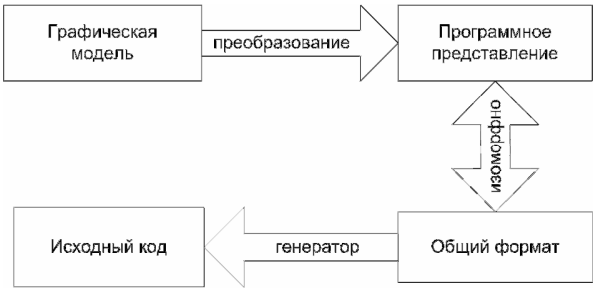


Рис. 1. Декомпозиция процесса генерации исходного кода

Разработанное инструментальное средство *MetaAuto* [6] построено по схеме, представленной на рис. 1.

2.2. Генерация кода

В рамках подхода к генерации исходного кода программы необходимо выполнить преобразование графа переходов автомата в исходный код программы. Вопрос об автоматической генерации кода с заданными свойствами является одним из вопросов решаемых в рамках порождающего программирования (*Generative Programming*) [7].

В настоящее время в рамках порождающего программирования разработано множество способов генерации кода. Среди них можно выделить:

- подстановки;
- подстановки с исполнением кода;
- обработчики данных регулярной структуры.

Генерация кода, основанная на подстановках, предполагает, что разработчик создает шаблон кода и набор данных в специальном формате, а затем с помощью вспомогательной программы выполняет подстановку этих данных в шаблон. Набор используемых в шаблоне подстановок может определяться как статически, так и динамически. Такой подход весьма нагляден и прост в использовании, однако имеет весьма ограниченную область применения и требует предварительной подготовки передаваемых для подстановки данных. Классический пример подстановок (с некоторыми оговорками) – шаблоны (templates) языка *C++*.

Гораздо шире возможности при генерации кода с использованием подстановки с исполнением кода. Этот вид генерации отличается от предыдущего возможностью применять в шаблоне не только подстановки, но также вставки исполняемого кода, оперирующего переданными в шаблон данными. Исполняемый код чаще всего использует язык, который специально создан для конкретного типа шаблонов и включает основные алгоритмические конструкции, такие как, например, простое ветвление (IF), выполнение итераций по переданным в качестве параметров спискам (WHILE), циклы с заданным количеством итераций (FOR).

Отметим, что в процессе усложнения конструкций, применяемых в шаблоне, и увеличения возможностей языка, уменьшается наглядность самих шаблонов.

Пример использования такого способа генерации приведен в работе [8]. В ней применяется технологии *ASP*, первоначально разработанная для генерации *HTML*-страниц. В качестве языка исполняемого кода в этой технологии используется язык *Visual Basic*.

Недостатком подходов к генерации кода, основанных на подстановках и подстановках с исполнением кода, является необходимость специальной подготовки данных для передачи в шаблон.

Третий способ генерации кода основан на обработчиках данных регулярной структуры и предполагает полное разделение данных и их представления. В этом случае шаблон играет роль обработчика данных и пишется на специальном метаязыке. Примером такого подхода является *XSLT*-обработка данных, представленных в *XML*-формате.

Вопрос генерации кода на основе *XSLT*-преобразований рассматривается, например, в работе [9].

Основным достоинством этого способа генерации кода является возможность обработки данных сложной структуры без предварительной подготовки этих данных.

Существование перечисленных видов генерации кода обусловлено неоднородностью решаемых в рамках порождающего программирования задач и структуры метаданных. Вопрос об использовании того или иного способа генерации зависит от условий конкретной задачи и данных, используемых для такой генерации. Рассмотрим особенности генерации исходного кода для графов переходов.

Графы переходов автоматов имеют большое количество рекурсивных структур: вложенные состояния, групповые переходы. Также генерация кода программы предполагает реализацию логических выражений, которая сильно отличается для языков высокого уровня и, например, для языка *Turbo Assembler*. Перечисленные особенности, а также и другие черты графов переходов определяют выбор способа генерации исходного кода – использование *XML*-представления графа переходов автомата и *XSLT*-преобразования для генерации исходного кода представляются наиболее удобными. Инструментальное средство *MetaAuto* использует *XSLT*-технологию для генерации исходного кода.

XSLT (*eXtensible Stylesheet Language Transformations* – расширяемый язык стилей для преобразований) в последнее время стал популярной *XML*-технологией. Спецификация определяет *XSLT* как язык для преобразований одних *XML*-документов в другие *XML*-документы. Однако за время своего существования *XSLT* стал использоваться значительно шире, например, для автоматической генерации кода.

В работе [9] рассматривается вопрос применения языка *XSLT* в качестве инструмента для генерации кода программ. Отметим основные достоинства и недостатки генерации кода с использованием языка *XSLT*.

XSLT-генерация кода имеет следующие достоинства:

- широкие возможности по изменению шаблонов, без изменения любой другой функциональности;
- широкие возможности по манипулированию данными. Возможность извлекать данные из дополнительных источников;
- возможность изменять язык программирования с помощью небольшого изменения шаблона;
- наглядность и широкое распространения *XML*-формата. Нет необходимости каждый раз придумывать новый формат хранения данных.

Недостатки языка *XSLT* при использовании его для генерации кода:

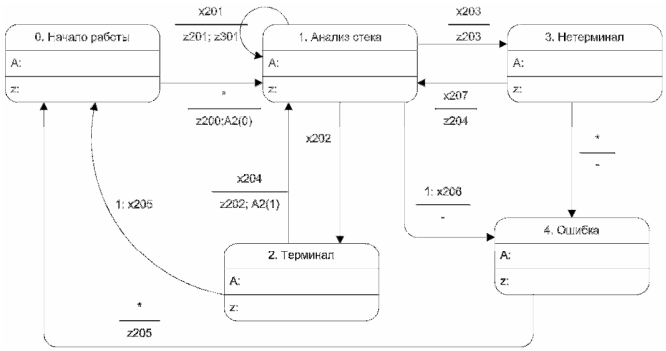
- трудно контролировать отступы и пробелы при генерации текста. Небольшое количество функций работы со строками. Спецификация *XSLT 2.0* призвана исправить эти недостатки;
- язык *XSLT* не предоставляет прямого доступа к операционной системе. Это несколько ограничивает возможности по генерации платформенно-специфичного кода;
- генерация нескольких исходящих документов в рамках спецификации *XSLT 1.0* невозможна.

3. Методика генерации исходного кода

Опишем методику генерации исходного кода с применением разработанного инструментального средства. Будем преобразовывать граф переходов из работы [10] в исходный код программы на языке *C#*.

3.1. Изображение графов переходов

Изобразим этот граф переходов с помощью редактора *MS Visio*. Расположим его на странице *A1* (рис. 2) файла `analizers.vsd`.



z200	Инициализация стека	x201	На вершине стека - команда
z201	Вынуть команду из стека	x202	На вершине стека - терминал
z202	Определить терминал стека	x203	На вершине стека - нетерминал
z203	Определить правило вывода по таблице разбора	x204	Терминал на вершине стека совпадает с текущей лексемой
z204	Поместить в стек правило вывода	x205	Текущая лексема - $ и вершина стека $
z205	Вывести сообщение об ошибке	x206	Текущая лексема - ошибка
z301	Исполнить команду стековой машины	x207	Текущее правило вывода существует (не ошибка)

Автомат	
Имя:	A1
Название:	Синтаксически анализатор

Рис. 2. Граф переходов автомата (страница *A1* файла `analizers.vsd`)

Для изображения графа переходов следует использовать специально созданный шаблон, содержащий все используемые на графе переходов типы элементов. Этот шаблон предоставляется вместе с инструментальным средством *MetaAuto*. Он включает в себя такие элементы, как «состояние», «переход», «описание автомата» и т.д.

3.2. Преобразование в *XML*-формат

Изображенный на первом шаге граф переходов следует преобразовать в *XML*-формат. Этот шаг включает в себя сразу два этапа (рис. 1): преобразование графического представления графа переходов в программное

представление и изоморфное преобразование программного представления в *XML*-формат.

Для такого преобразования воспользуемся компонентой `Visio2Xml.exe`, являющейся частью инструментального средства. В качестве параметров передадим компоненте путь до файла, содержащего графическую модель – `analizers.vsd` и название итогового *XML*-файла. Для запуска из командной сроки используем следующую командную строку:

```
Visio2Xml.exe analizers.vsd
automatas.xml
```

Полученный *XML*-файл полностью описывает граф переходов. Этот файл, а также подробное описание его формата, приведены в работе [6].

3.3. Создание *XSLT*-шаблона

Для получения кода на языке C# необходимо создать соответствующий шаблон. Как правило, по одному графу переходов требуется построить сразу несколько файлов. Например, в языке *C++* требуется создать файл заголовков и файл реализации. В таком случае необходимо создать шаблон для каждого такого файла.

Назовем шаблон для нашего примера `automatas.cs.xslt`.

Описание шаблонов *XSLT* приведено в работе [6]. Выполняемые в шаблонах действия весьма стандартны. Примеры создания таких шаблонов можно найти практически в любом пособии по языку *XSLT*.

3.4. Получение исходного кода

Преобразуем полученный на втором шаге *XML*-файл в исходный код с помощью *XSLT*-шаблона, созданного на предыдущем шаге. Воспользуемся компонентой `XslTransform.exe` инструментального средства *MetaAuto*. Данной компоненте необходимо передать в качестве параметров путь до *XML*-файла (`automatas.xml`), путь до шаблона (`automatas.cs.xslt`) и название итогового файла с исходным кодом (`analyzers.cs`).

Для запуска компоненты воспользуемся следующей командной строкой:

```
XslTransform.exe automatas.xml
automatas.cs.xslt analyzers.cs
```

В результате получим файл `analyzers.cs`, содержащий исходный код на языке *C#*.

4. Заключение

В статье описан подход к автоматической генерации исходного кода программ с явным выделением состояний. Рассмотрены различные технологии и методики, описаны их преимущества и недостатки. Также описана реализация предложенного подхода в инструментальном средстве *MetaAuto*.

Это инструментальное средство разрабатывалось как генератор кода программ с явным выделением состояний на любом априори заданном языке программирования. Оно было использовано для генерации кода синтаксического и лексического анализаторов, использующихся самим инструментальным средством. Таким образом, первым успешным внедрением инструментального средства стало оно само.

Однако, после написания прототипа инструментального средства, оказалось, что его также можно использовать и для верификации программ. Такая верификация было выполнена в работе [11]. С помощью инструментального средства процесс создания верификатора был достаточно прост, так как граф переходов легко доступен при использовании программного представления модели.

Для оценки эффективности предлагаемого подхода в качестве примера было выполнено сравнение традиционного и предлагаемого подходов к созданию ПО для кнопочного телефона [11]. Время, затраченное в том, и в другом случае, было примерно одинаковым. Однако в первом случае отладка и тестирование заняли большую часть времени, а во втором – программа проектировалась, и по проекту генерировался код, который практически сразу работал. "В сухом остатке" во втором случае осталась проектная документация, по которой можно понять структуру и поведение программы даже через несколько лет после ее разработки, что весьма трудно при традиционном подходе.

Авторы надеются, что конфигурируемость инструментального средства и простота его использования позволит более эффективно внедрять программирование с явным выделением состояний.

Изложенный подход предполагается адаптировать применительно к разработке программного обеспечения встроенных систем.

Литература

[1] *Новиков Ф.А.* Визуальное конструирование программ //Информационно-управляющие системы. 2005. № 6, с.9–22. http://is.ifmo.ru/works/visualcons

[2] *Шалыто А.А.* Технология автоматного программирования //Мир ПК. 2003. № 10, с.74–78. http://is.ifmo.ru/works/tech_aut_prog/

[3] *Finite state machine. Wikipedia.* The free encyclopedia. http://en.wikipedia.org/wiki/Finite_automaton

[4] *Головешин А.* Использование конвертора Visio2SWITCH, 22 с. http://is.ifmo.ru/progeny/visio2switch

[5] *Гуров В., Мазин М., Нарвский А., Шалыто А.* UML. SWITCH-технология. Eclipse //Информационно-управляющие системы. 2004. № 6, с.9–22. http://is.ifmo.ru/works/uml-switch-eclipse

[6] *Канжелев С., Шалыто А.* Преобразование графов переходов, представленных в формате *MS Visio*, в исходные коды программ для различных языков

программирования (инструментальное средство MetaAuto), 102 с. http://is.ifmo.ru/projects/metaauto

[7] *Чарнецки К., Айзенекер У.* Порождающее программирование: методы, инструменты, применение. СПб.: Питер, 2005, 736 с.

[8] *Селлз К.* Современные способы автоматизации повторяющихся задач программирования //MSDN Magazine. 2002, № 6.

[9] *Dodds L.* Code generation using *XSLT*. http://www-106.ibm.com/developerworks/edu/x-dw-codexslt-i.html

[10] *Шалыто А., Туккель Н.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5, с.45–62 http://is.ifmo.ru/works/switch

[11] *Канжелев С., Шалыто А.* Моделирование кнопочного телефона с использованием SWITCH-технологии. Вариант 2. 104 с. http://is.ifmo.ru/projects/phone