УДК 681.3.06

ПРЕДМЕТНО-ОРИЕНТИРОВАННЫЙ ЯЗЫК АВТОМАТНОГО ПРОГРАММИРОВАНИЯ НА БАЗЕ ДИНАМИЧЕСКОГО ЯЗЫКА RUBY

О. Г. Степанов,
программист
Компания JetBrains s. r. o.
А. А. Шалыто,
доктор техн. наук, профессор
Д. Г. Шопырин,
канд. техн. наук, доцент
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Решается задача преобразования диаграмм переходов, применяемых в SWITCH-технологии, в исполняемый код. Предлагается использование динамических языков программирования, позволяющих добиться изоморфности диаграммы и соответствующего программного кода, что ведет к уменьшению количества ошибок при указанном преобразовании. На базе динамического языка Ruby с использованием разработанной библиотеки STROBE создан предметно-ориентированный язык автоматного программирования.

This paper solves the problem of transforming transition diagrams used in SWITCH technology into executable code. We suggest using dynamic programming languages to solve this problem because these languages allow to achieve an isomorphism between the source diagram and the resulting code. This leads to decreasing the number of errors that appear during such transition. We also present an extension library STROBE and a domain-specific automata programming language based on the dynamic programming language Ruby.

Введение

Для проектирования и разработки реактивных систем часто используется SWITCH-технология, поддерживающая «автоматное программирование» или «программирование с явным выделением состояний» [1]. Одной из основных частей этой технологии является графический язык, позволяющий описать поведение различных подсистем в терминах состояний и переходов между ними и связей между этими системами в виде диаграмм переходов и связей соответственно.

При использовании SWITCH-технологии в разработке программного обеспечения важной частью является реализация поведения, описанного диаграммами переходов, на целевом языке программирования. Особенностью диаграмм, используемых в указанной технологии, являются сложные условия, образуемые символами входных воздействий (событий и входных переменных), и наличие символов выходных воздействий в вершинах и/или на переходах. Автоматы, соответствующ таким диаграммам переходов, в теории автомат называются структурными. Еще одна особенност применяемых диаграмм переходов состоит в ток что в их вершинах могут быть указаны символ вложенных в них автоматов.

Для решения этой задачи традиционно исполь зуется один из трех подходов.

1. Полностью ручное программирование. Одни из простейших общепринятых методов такого пр граммирования является следующий: текущее со стояние системы хранится в переменной инте рального или перечислимого типа и основная л гика программы сосредоточена внутри одного и нескольких операторов `switch`, определяющ действия программы в зависимости от текуще состояния [2]. Другим методом является испол зование паттерна программирования `State` [3 Несмотря на такие достоинства этой группы мет дов как высокая производительность и *полны*

контроль над получаемым кодом, она обладает существенными недостатками: низкой читаемостью кода и большой трудоемкостью.

2. Автоматическая генерация кода по диаграмме переходов. Обычно при таком подходе генерируется код, аналогичный с получаемым при использовании ручного программирования. Недостатками этого подхода являются:

• низкая читаемость кода, связанная с тем, что в качестве целевого используется императивный язык, например Java;

• потеря информации, специфичной для логики диаграмм переходов (вершины диаграммы и ее переходы заменяются их образами на целевом языке — классами и кодом выполнения переходов);

• малая степень контроля над получаемым кодом и невозможность ручного изменения этого кода;

• привязанность к конкретному формату входных данных, с использованием которого задается исходная диаграмма переходов (например, формат файлов Visio [4], XML [5, 6]).

3. Ручное написание кода с применением специальной библиотеки. В этом случае происходит перенос диаграммы переходов в вызовы указанной библиотеки, которая по этим инструкциям строит внутреннее представление рассматриваемой диаграммы. Затем по этому представлению происходит реализация автомата. Основным преимуществом этого подхода является то, что вызовы библиотеки отражают семантику диаграммы переходов (каждый вызов может, например, соответствовать объявлению состояния или перехода). Это позволяет создавать читаемый код, который легко поддерживать. Также некоторые библиотеки, указанные ниже, ориентированы на конкретные виды взаимодействия автоматного и объектно-ориентированного кода, что позволяет более эффективно объединять эти подходы к программированию. Основными недостатками такого подхода являются:

• низкая производительность некоторых библиотек (существуют, однако, реализации, основанные на метапрограммировании и статической генерации кода [7, 8], которые позволяют повысить производительность);

• невозможность описать ряд конструкций диаграмм переходов, используя ограниченный синтаксис целевого языка.

В настоящей работе предлагается развитие третьего подхода за счет повышения «качества» и читаемости получаемого кода. Разработанный подход использует динамические языки программирования — языки, которые позволяют изменять и дополнять код программы во время выполнения. В работе показано, как свойства динамических языков могут быть использованы для увеличения читаемости кода, генерирующего модель системы, особенно в части формирования условий переходов.

Поясним изложенное. В рамках настоящей работы предлагается:

• разработать *текстовый язык автоматного программирования*, который реализован на языке Ruby. Этот язык относится к классу предметно-ориентированных языков программирования (Domain Specific Language — DSL [9]);

• по заданной диаграмме переходов, описывающей поведение автомата, вручную строится его описание на разработанном текстовом языке;

• это текстовое описание транслируется. В результате трансляции с помощью разработанной в настоящей работе библиотеки STROBE (поддерживает такие понятия, как «состояние», «переход», «событие», «входное воздействие», «выходное воздействие» и т. д.) в памяти строится модель диаграммы переходов;

• построенная модель является исполняемой и используется для обработки входных воздействий.

Для формализации процесса переноса диаграмм переходов в код на предлагаемом текстовом языке была разработана операционная семантика (правила интерпретации) диаграмм переходов структурных автоматов. Предложенная семантика обеспечивает корректность реализации диаграмм переходов.

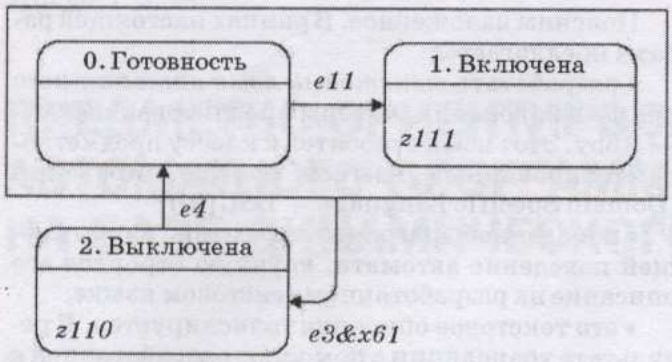
Статья имеет следующую структуру: в первом разделе предложена операционная семантика диаграмм переходов и описана проблема их переноса в исполняемый код. Во втором разделе предложен подход, позволяющий использовать свойства динамических языков для решения задачи переноса диаграмм переходов. В третьем разделе описана библиотека STROBE, реализующая разработанный подход.

Операционная семантика диаграмм переходов

При реализации поведения, описанного с помощью диаграмм переходов на языках программирования, возникают две проблемы, первая из которых состоит в корректном переносе поведения системы в исполняемый код, а вторая — в сохранении исходных обозначений и структуры описания автомата.

Рассмотрим пример использования диаграмм переходов. В работе [10] описана система управления лифтом, которая содержит автомат управления кнопкой вызова лифта на нижнем этаже (*A11*). Его диаграмма переходов представлена на рисунке.

Этот автомат имеет три состояния: «Готовность» (кнопка на этаже подсвечена и ее можно нажать), «Включена» (кнопка нажата) и «Выключена» (кнопку нажать нельзя). Переход в состояние «Включена» происходит по нажатию кнопки (событие *e11*), переход в состояние «Готовность» — по событию от головного автомата *e4* (разрешение на включение лампы в кнопке),



■ *Диаграмма переходов автомата управления кнопкой вызова лифта*

переход в состояние «Выключена» — по событию от головного автомата *e3* (выключение лампы в кнопке) при условии, что лифт находится на нижнем этаже (определяется переменной *x61*).

Для корректного преобразования диаграмм переходов в исполняемый код предложим операционную семантику и метод для выполнения указанного преобразования. Эта семантика в некотором смысле аналогична семантике UML-диаграмм состояний, описанной в работах [11, 12].

Определим основные свойства диаграмм переходов.

1. Диаграмма изображает одно или несколько состояний системы и переходы между ними.

2. Состояния на диаграмме могут быть объединены в группы, которые могут быть вложены друг в друга. Состояния внутри группы равноправны.

3. Переходы могут начинаться в состоянии или в группе состояний, а заканчиваться только в состоянии (переходы, начинающиеся в группе состояний, называются *групповыми переходами*). Переходы могут начинаться и заканчиваться в одном и том же состоянии. При этом они называются *петлями*.

4. Каждое состояние помечено следующими атрибутами:

- имя состояния;
- номер состояния (нумерация начинается с нуля);
- действия при входе в состояние;
- вложенные автоматы (возможно с номерами воздействий, с которыми они вызываются).

5. Переход может быть помечен следующими атрибутами:

- условие перехода;
- действия на переходе;
- приоритет перехода.

Обработка события происходит следующим образом: перебираются переходы, выходящие из текущего состояния и содержащие это состояние групп в порядке приоритета (сначала рассматриваются переходы с меньшими номерами). Для каждого перехода вычисляется условие, которое является булевой формулой. Эта формула может использовать следующие символы:

- *ei* — имеет значение «истина», если *i*-е событие наступило, и «ложь» — в противном случае (при написании программ событие *e0* используется для инициализации каждого автомата);
- *xi* — переменная *xi*;
- *yi* — номер состояния автомата *Ai*.

Выполняется первый переход, для которого значение условия истинно. Выполнение перехода состоит из следующих шагов:

- выполняются действия на переходе (вызываются выходные воздействия *zi* в порядке их следования на переходе);
- осуществляется переход в состояние, в котором заканчивается переход;
- если произошла смена состояния, то выполняются действия при входе в состояние;
- вызываются вложенные автоматы: если для автомата указан номер события, он вызывается с этим событием, иначе – с событием *e0*.

Применим описанную семантику к автомату *A11*.

1. Если текущее состояние — «Готовность» и наступило событие *e11*, то выполняется переход в состояние «Включена». При этом производится выходное воздействие *z111* (включить лампу в кнопке).

2. Если текущее состояние — «Готовность» или «Включена», наступило событие *e3* и при этом значение переменной *x61* — «истина», то осуществляется переход в состояние «Выключена». При этом производится выходное воздействие *z110* (выключить лампу в кнопке).

3. Если текущее состояние — «Выключена» и наступило событие *e4*, то производится переход в состояние «Готовность».

Реализация автоматных систем на языке Ruby

Для решения задачи переноса диаграмм переходов в исполняемый код предлагается использовать динамические языки программирования [13]. Отличительными свойствами этих языков, позволяющими упростить перенос диаграмм переходов, являются:

- динамическая генерация кода;
- использование замыканий (*замыкание* — совокупность процедуры и связанного с ней лексического контекста, http://en.wikipedia.org/wiki/Closure_%28computer_science%29).

Динамическая генерация кода позволяет для каждой диаграммы переходов создавать индивидуальный код выполнения автомата. Это обеспечивает повышение производительности исполняемого кода.

Для переноса условий переходов в код на целевом языке программирования практически без изменений используются замыкания. Это позволяет сохранить естественную запись условий.

Для практической реализации предложенного подхода был выбран язык Ruby [13], разработан-

ный Юкихиро Мацумото (Yukihiro Matsumoto). Этому динамическому языку программирования присущ ряд полезных свойств:

- динамичность (возможность изменения и дополнения программы на лету);
- простой синтаксис;
- объектная ориентированность;
- поддержка примесей (http://ru.wikipedia.org/wiki/Mixin), позволяющих уточнить поведение создаваемых классов за счет включения модулей;
- поддержка замыканий;
- легкая переносимость (среда исполнения Ruby может быть запущена в большинстве операционных систем);
- возможность интеграции с кодом на других языках программирования.

В настоящей работе разработана библиотека STROBE и построенный на ее основе текстовый язык автоматного программирования. Они позволяют декларативно задавать диаграммы переходов на языке Ruby. Для программиста эта библиотека предоставляет дополнительный набор инструкций (реализованных в виде методов), позволяющих поэлементно определять диаграммы переходов (в число инструкций входят, например, `state` для определения состояния и `transition` для определения перехода). Методы используют технологию именованных параметров (при вызове метода значение каждого параметра связывается с параметром явно по имени). Это позволяет увеличить читаемость кода. Именами параметров, использованных в примере ниже, являются `:to` или `:output_actions`.

Для пояснения структуры разрабатываемого языка приведем код на этом языке, описывающий поведение автомата *A11*:

```
# Подключение библиотеки STROBE
require 'strobe/automaton'
module Elevator
  # Декларация класса автомата
  class A11 < Strobe::Automaton
    # Декларация внутренней переменной x61
    attr_accessor :x61
    # Декларация событий
    inputs :e3, :e4, :e11
    # Декларация выходных воздействий
    outputs :z110, :z111
    # Начало группы состояний
    begin_group
      # Состояние "Готовность"
      state :ready
        # Переход в состояние "Включена"
        transition :to => :on,
                   :if => lambda { e11 }
      # Состояние "Включена"
      state :on,
            :output_actions => :z111
      # Групповой переход в состояние "Выключена"
      group_transition :to => :off,
                       :if => lambda { e3 && x61 }
    end_group
    # Состояние "Выключена"
    state :off,
          :output_actions => :z110
      # Переход в состояние "Включена"
      transition :to => :ready,
                 :if => lambda { e4 }
  end
end
```

В этом примере декларирован класс *A11*. Он объявлен автоматным через наследование от библиотечного класса `Strobe::Automaton`. Внутри декларации класса последовательно определена диаграмма состояний автомата *A11*.

Инструкции `inputs`, `outputs`, `state`, `transition` и `group_transition` декларируют события, выходные воздействия, обычный и групповой переход соответственно. Инструкции `begin_group` и `end_group` объединяют все декларированные между ними состояния в группу.

Метод преобразования диаграмм переходов в исполняемый код на языке Ruby

Представленный выше пример был построен с использованием формального метода, позволяющего перенести любую диаграмму переходов в исполняемый код на языке Ruby.

Опишем схему метода.

1. Для каждой диаграммы создается новый класс с именем, совпадающим с названием автомата.

2. Внутри класса, как это было выполнено в примере, с помощью конструкций предлагаемого языка описывается диаграмма переходов. При этом:

- состояниям присваиваются идентификаторы, соответствующие их именам на диаграмме;
- в описаниях переходов ссылки на состояния производятся по именам.

3. Класс автомата связывается с неавтоматной частью программы через подписку на выходные воздействия автомата.

Ключевыми моментами использования языка Ruby являются полностью декларативное описание поведения системы и автоматическое порождение специальных методов, позволяющих переносить диаграмму переходов с минимумом изменений. Покажем это на примере. Допустим, что в диаграмме существует переход, условие которого задается формулой

$$e1 \vee e0 \wedge (x1 \vee (y2 = 3)).$$

В коде программы эта формула будет представлена в виде выражения

```
e1 || e0 && (x1 || (y2 == 3)).
```

При вычислении условия перехода это выражение будет вычислено в контексте автоматного класса, что позволит использовать порожденные по описанию автомата методы `e0`, `e1`, `x1` и `y2`. При этом подвыражение `y2 == 3` принимает значение «истина», если второй автомат находится в

третьем состоянии, и значение «ложь» — в противном случае.

В общем случае по описанию автомата порождаются следующие методы:

• для каждого события — метод с именем события, который возвращает значение «истина», если наступило это событие, и «ложь» — в противном случае;

• для каждой входной переменной — метод с именем переменной, который возвращает текущее ее значение;

• для связей с другим автоматом системы — метод, имя которого состоит из символа `y` и номера автомата. Этот метод возвращает номер текущего состояния этого автомата.

При попытке практической реализации предложенного подхода возникает ряд проблем:

• возможность использования конструкций описания автомата непосредственно в теле класса наряду со стандартными декларациями языка;

• обеспечение выполнения условий переходов в соответствующем контексте;

• поддержка наличия нескольких экземпляров одного и того же автомата и обеспечение связей между автоматами через переменные `yi`;

• поддержка интеграции с программами на других языках, включая обеспечение возможности управления физическими устройствами;

• обеспечение приемлемой производительности получаемого кода.

Поставленные проблемы решены в библиотеке STROBE, особенности реализации которой описаны ниже.

Реализация библиотеки STROBE

Библиотека STROBE позволяет перенести в код на языке Ruby любую синтаксически верную диаграмму переходов, а также перенести несколько автоматов и связать их. Возможна также интеграция с модулями на других языках программирования, в том числе модулями управления физическими объектами. При этом описание автомата изоморфно диаграмме состояний и понятно без дополнительных инструкций и описаний.

Описание автомата с помощью библиотеки STROBE имеет следующие особенности.

1. Внешняя конфигурация автомата (входные и выходные воздействия, связи с другими автоматами) описывается в начале класса.

2. Состояния описываются последовательно, в порядке их нумерации на диаграмме переходов.

3. Переходы из состояния описываются непосредственно после описания самого состояния.

4. Связь с другими компонентами осуществляется путем подписки на конкретные выходные воздействия.

5. Связь с другими автоматами по переменным состояния (`yi`) выполняется путем явного связывания экземпляров автомата и имен переменных в блоке описания конфигурации автомата.

6. Для независимого выполнения нескольких экземпляров одного автомата используется система доменов.

Домены — логические области, к одной из которых может быть приписан экземпляр автомата при его создании. Каждый экземпляр автомата имеет уникальный в пределах домена идентификатор, по умолчанию равный имени класса. Таким образом, ссылаться на другие автоматы можно по имени класса, используя стандартные алгоритмы разрешения ссылок языка.

Производительность решения находится на одном уровне с динамически выполняемыми решениями третьей группы (интерпретирующими внутреннее представление диаграммы переходов). Однако по всем остальным показателям предложенный подход решает проблемы, описанные в конце предыдущего раздела.

Заключение

В настоящей работе поставлена и решена проблема переноса диаграмм переходов автоматов, разработанных по SWITCH-технологии, в исполняемый код. Были рассмотрены основные направления решения этой проблемы и предложен подход, развивающий одно из них. Этот подход заключается в создании текстового предметно-ориентированного языка описания автоматов (более подробно он описан в работе [14]) на базе динамического языка Ruby. Описаны особенности этого подхода: декларативная структура кода и его изоморфность (особенно в области задания условий переходов) исходной диаграмме. Описаны основные проблемы, возникающие при практической реализации рассматриваемого подхода, и предложена конкретная реализация на динамическом языке программирования Ruby, решающая большинство этих проблем.

Литература

1. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с. http://is.ifmo.ru/books/switch/1
2. **Шалыто А. А., Туккель Н. И.** Реализация автоматов при программировании событийных систем // Программист. 2002. № 4. С.74–80. http://is.ifmo.ru/works/evsys/
3. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001. 368 с.

4. **Инструментальное средство** автоматного программирования Visio2SWITCH. http://is.ifmo.ru/progeny/visio2switch/
5. **State Chart XML (SCXML)**: State Machine Notation for Control Abstraction 1.0. http://www.w3.org/TR/2005/WD-scxml-20050705/
6. **Инструментальное средство** автоматного программирования UniMod. http://unimod.sourceforge.net
7. **The Boost C++ Metaprogramming Library**. www.mywikinet.com/mpl/paper/mpl_paper.pdf
8. **Шопырин Д. Г., Шалыто А. А.** Объектно-ориентированный подход к автоматному программированию // Информационно-управляющие системы. 2003. № 5. С. 29–39. http://is.ifmo.ru/works/ooaut/
9. **Чарнецки К., Айзенкер У.** Порождающее программирование. Методы, инструменты, применение. СПб.: Питер, 2005.
10. **Наумов А. С., Шалыто А. А.** Система управления лифтом. Проектная программная документация / СПбГУ ИТМО. СПб., 2003. http://is.ifmo.ru/projects/elevator/
11. **Гуров В. С., Мазин М. А., Шалыто А. А.** Операционная семантика UML-диаграмм состояний в программном пакете Unimod // Материалы научно-метод. конф. «Телематика-2005» / СПбГУ ИТМО. СПб., 2005. http://tm.ifmo.ru/tm2005/src/224as.pdf
12. **Маврин П. Ю.** Реализация диаграмм состояний / СПбГУ ИТМО. СПб., 2006. http://is.ifmo.ru/papers/statec/
13. **Thomas D., Fowler C., Hunt A.** Programming Ruby. Second Edition. Pragmatic Bookshelf, 2004.
14. **Степанов О. Г.** Автоматное программирование с использованием динамических языков программирования: Магистерская дис. / СПбГУ ИТМО. СПб., 2006. http://is.ifmo.ru, раздел «Работы».