УДК 681.3.06

НАСЛЕДОВАНИЕ АВТОМАТНЫХ КЛАССОВ С ИСПОЛЬЗОВАНИЕМ ДИНАМИЧЕСКИХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ (НА ПРИМЕРЕ ЯЗЫКА RUBY)

К. И. Тимофеев,
системный архитектор
компания «DataArt»
А. А. Астафуров,
аспирант
А. А. Шалыто,
доктор техн. наук, профессор
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Рассматриваются достоинства применения динамических языков программирования для реализации наследования автоматных классов. Предлагаемый подход иллюстрируется примером реализации функциональности регистрации пользователей на сайте.

Ключевые слова *— объектно-автоматное программирование, наследование автоматных классов, динамические языки программирования.*

Введение

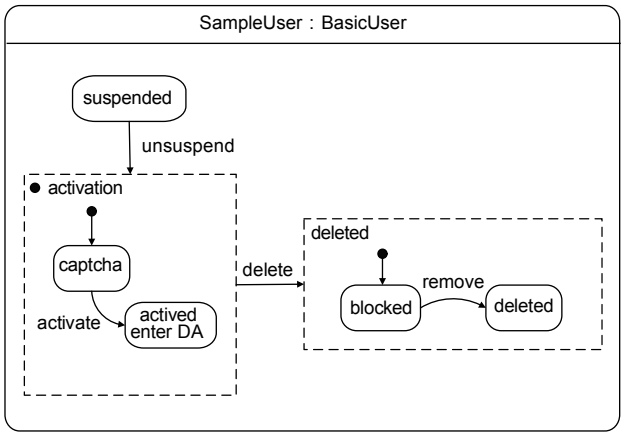
С каждым годом динамические языки программирования все шире используются для разработки программного обеспечения. Так, в 2008 г. отношение динамических языков к языкам со статической типизацией составило около 40 % [1]. Динамические языки программирования позволяют расширять программу во время ее выполнения за счет создания новых методов и классов, а также использования макросов [2]. Язык Ruby является динамическим и входит в десятку самых популярных языков программирования 2008 г. [1]. На нем, в частности, разрабатывается фреймворк для создания веб-приложений Ruby on Rails [3], который является Open Source-проектом. Его особенность — наличие гибкого механизма расширения функциональности с помощью плагинов (компонентов, которые можно добавить в приложение Ruby on Rails). Одним из таких плагинов является Restful-authentication [4], расширяющий веб-приложение функциональностью регистрации пользователей. Этот плагин применяется в 96 % [5] приложений Ruby on Rails и реализован с использованием конечных автоматов.

Рассматриваемый плагин использует библиотеку Acts as State Machine [6], которая не позволяет сохранить иерархию родительского автомата при наследовании и не имеет вложенных групп состояний. Это приводит к дублированию кода, затрудняет отладку, а также делает невозможным изоморфное построение модели по коду при необходимости. В работе [2] был предложен метод наследования автоматных классов, для поддержки которых разработана библиотека State Machine on Steroids. Она использует расширенную динамическими свойствами объектно-ориентированную парадигму, суть которой состоит в том, что автоматные классы создаются во время работы программы. Использование этого метода, по мнению авторов, позволит устранить указанные выше недостатки реализации плагина Restful-authentication.

Цель настоящей работы — сравнение указанных подходов к наследованию автоматных классов на примере плагина Restful-authentication.

Используемая графическая нотация

В качестве графической модели предлагается использовать диаграмму состояний UML 2 [7],



■ ***Рис. 1.*** *Расширенная графическая нотация*

расширенную графической нотацией для наследования автоматных классов, предложенной в работе [8], так как в UML 2 невозможно отобразить наследование автоматов.

В примере использования расширенной графической нотации (рис. 1) отображен автоматный класс SampleUser, унаследованный от родительского автоматного класса BasicUser, который содержит в себе следующие изменения:

• добавлена новая группа deleted, состоящая из двух состояний blocked, deleted. Состояние blocked является начальным;

• переопределена группа activation — начальным состоянием является captcha. Добавлен новый переход из этой группы в новую группу deleted;

• добавлено новое состояние suspended, которое имеет переход на группу activation.

Описание проблемы

Рассмотрим плагин Restful-authentication подробнее. Пусть, например, для успешной регистрации пользователь должен выполнить три действия:

1) заполнить регистрационную форму на сайте;

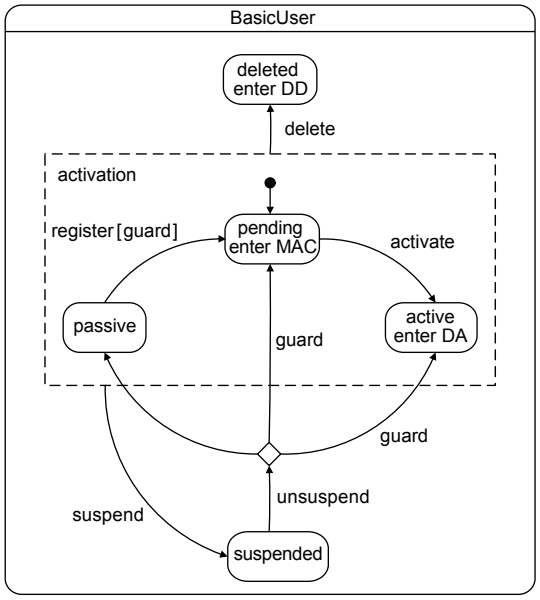
2) получить письмо с кодом активации;

3) подтвердить регистрацию, введя полученный код в специальном разделе сайта.

В случае, если пользователь не зарегистрируется в течение заданного времени или несколько раз неверно введет код активации, он будет заблокирован и не сможет больше зарегистрироваться на сайте.

Администратор может как разблокировать пользователя, так и удалить его из системы.

Автомат регистрации базового пользователя (BasicUser), представленный на рис. 2 с помощью расширенной графической нотации, состоит из группы activation и состояний suspended и deleted. Вложенная группа activation включает три состояния: passive, active и pending, последнее из которых является начальным.



■ ***Рис. 2.*** *Автомат регистрации базового пользователя*

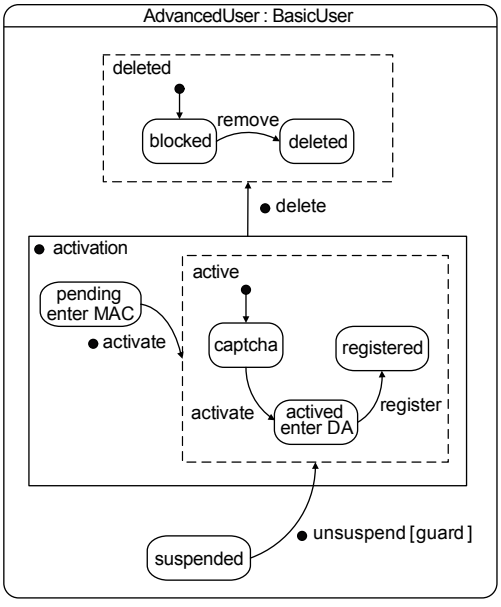
В этом автомате определены следующие переходы:

delete — удалить пользователя из системы;

suspend — перевести пользователя в состояние ожидания в случае следующих событий:

• пользователь не активировал аккаунт в течение семи дней с момента регистрации;

• пользователь ввел три раза неверный код активации;



■ ***Рис. 3.*** *Автомат регистрации расширенного пользователя*

register — пользователь ввел регистрационные данные, однако еще не подтвердил регистрацию;

activate — пользователь ввел регистрационные данные и успешно подтвердил активацию;

unsuspend — администратор может предоставить пользователю возможность повторно зарегистрироваться.

Автомат AdvancedUser (рис. 3) унаследован от автомата BasicUser и обладает следующей функциональностью.

1. Пользователь после ввода кода активации должен совершить два дополнительных действия: корректно распознать Captcha (для удаления нежелательных автоматических регистраций) и заполнить данные в пользовательском разделе.

Для этого создадим вложенную группу active.

2. Удаление пользователя проходит в два этапа: на первом пользователь блокируется, однако его сообщения остаются в системе, а на втором из системы удаляется не только пользователь, но и все его данные.

Для этого создадим новую вложенную группу deleted.

Реализация наследования с использованием библиотеки Acts as State Machine

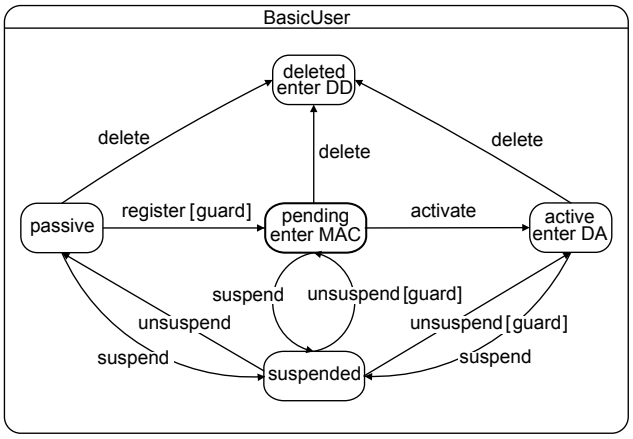
Рассмотрим класс базового пользователя (BasicUser) в плагине Restful-authentication с использованием библиотеки Acts as State Machine. Она предоставляет четыре метода:

acts_as_state_machine :initial — устанавливает начальное состояние автомата;

state — создает состояние. В качестве необязательных аргументов принимает лямбда-функции [2] — действия при входе в состояние и при выходе из него;

event — именованный переход;

transitions — определяет, из какого и в какое состояние будет осуществлен именованный переход. В качестве необязательного аргумента метод принимает лямбда-функцию — условие, при котором осуществляется переход.



■ ***Рис. 4.*** *Автомат регистрации базового пользователя без вложенных групп*

Эта библиотека не поддерживает вложенные группы. Поэтому для изоморфного переноса требуется модифицировать автомат BasicUser, как показано на рис. 4.

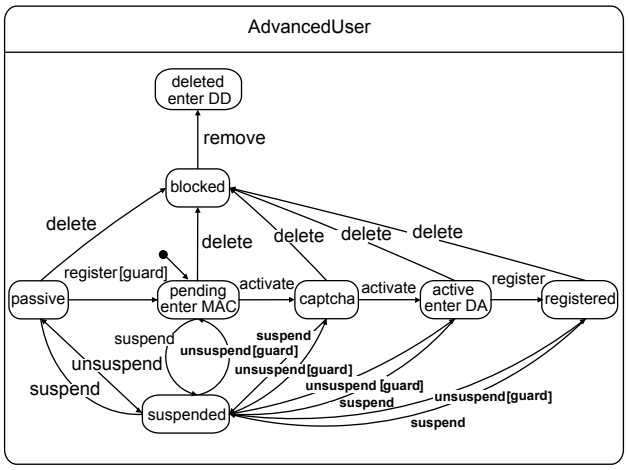
Ниже приведен код на языке Ruby, который реализует эту модель:

```
class BasicUser
  acts_as_state_machine   :initial =>   :pending

  state   :passive
  state   :pending,   :enter =>   :make_activation_code
  state   :active,    :enter =>   :do_activate
  state   :suspended
  state   :deleted,   :enter =>   :do_delete

  event   :register do
    transitions   :from =>   :passive,   :to =>   :pending,
                  :guard => Proc.new {|u|
!(u.crypted_password.blank? && u.password.blank?) }
  end

  event   :activate do
    transitions   :from =>   :pending,   :to =>   :active
  end

  event   :suspend do
    transitions   :from => [:passive,   :pending,   :active],
:to =>   :suspended
  end

  event   :delete do
    transitions   :from => [:passive,   :pending,   :active,
:suspended],   :to =>   :deleted
  end

  event   :unsuspend do
    transitions   :from =>   :suspended,   :to =>   :active,
                  :guard => Proc.new {|u|
!u.activated_at.blank? }
    transitions   :from =>   :suspended,   :to =>   :pending,
                  :guard => Proc.new {|u|
!u.activation_code.blank? }
    transitions   :from =>   :suspended,   :to =>   :passive
  end
end
```

Создадим новый класс AdvancedUser (рис. 5), который будет наследован от класса BasicUser. Он наследует всю внутреннюю структуру класса BasicUser, созданную методами библиотеки Acts as State Machine. Так как эта библиотека не поддерживает вложенные группы, то иерархия родительского автомата BasicUser в классе AdvancedUser будет потеряна.

Ниже приведен код, реализующий эту модель:

```
class AdvancedUser < BasicUser
  state   :captcha
  state   :registered
  state   :blocked
```



■ ***Рис. 5.*** *Модель автомата в классе* AdvancedUser *после потери иерархии родительского автомата*

```
  event   :activate do
    transitions   :from =>   :captcha,   :to =>   :actived
    transitions   :from =>   :pending,   :to =>   :captcha
  end

  event   :register do
    transitions   :from =>   :activated,   :to =>   :registered
  end

  event   :unsuspend do
    transitions   :from =>   :unsuspend,   :to =>   :captcha
  end

  event   :suspend do
    transitions   :from => [:passive,   :pending,   :active,
:suspended,   :captcha,   :registered],   :to =>   :suspended
  end

  event   :delete do
    transitions   :from => [:passive,   :pending,   :active,
:suspended,   :captcha,   :registered],   :to =>   :deleted
  end
end
```

Недостатками этого решения являются:

— потеря иерархии родительского автомата, при этом автоматический изоморфный перенос кода в модель невозможен;

— отсутствие вложенных групп состояний, что приводит к увеличению необходимого числа переходов.

Реализация наследования с использованием библиотеки State Machine on Steroids

Для устранения недостатков предыдущего решения авторами разработана библиотека State Machine on Steroids. Она использует динамические языки программирования для реализации классов автомата. Библиотека предоставляет шесть методов, из которых четыре аналогичны предоставляемым библиотекой Acts as State Machine, а также:

automaton — метод принимает в качестве аргумента блок, для которого определен DSL-синтаксис (Domain Specific Language) автомата [2];

group — создание вложенной группы состояний.

Реализуем автомат, представленный на рис. 2:

```
class BasicUser
  include StateMachineOnSteroids

  automaton   :user,   :initial =>   :activation do
    state   :deleted

    group   :activation, initial =>   :pending do
      state   :pending do
        transition   :activate,   :to =>   :actived
      end

      state   :passive do
        transition   :register,   :to =>   :pending,
                     :guard => Proc.new {|u|
!(u.crypted_password.blank? && u.password.blank?) }
      end

      state   :activated

      transition   :suspend,   :to =>   :suspended
      transition   :delete,   :to =>   :deleted
    end

    state   :suspended do
      event   :unsuspend do
        transition   :to =>   :active,
                     :guard => Proc.new {|u|
!u.activated_at.blank? }
        transition   :to =>   :pending,
                     :guard => Proc.new {|u|
!u.activation_code.blank? }

        transition   :to =>   :passive
      end
    end
  end
end
```

В этом коде каждый вызов метода библиотеки State Machine on Steroids создает класс. Благодаря этому при наследовании сохраняется структура родительского автомата BasicUser.

Реализуем наследованный класс AdvancedUser (см. рис. 3):

```
class AdvancedUser < BasicUser
  automaton   :user do
    group   :deleted, initial =>   :blocked do
      state   :blocked do
        transition   :remove,   :to =>   :deleted
      end

      state   :user::activation::deleted
    end

    group   :activation do
      group   :active, initial =>   :captcha do
        state   :captcha do
          transition   :activate,   :to =>   :activated
```

```
        end

        state   :user::activated do
          transition   :register,   :to =>   :registered
        end

        state   :registered
      end
    end

    transition   :user::delete,   :to =>   :deleted
    transition   :user::activate,   :to =>   :actived
    state   :suspended do
      event   :user::unsuspend,   :to =>   :active
    end
  end
end
```

При использовании библиотеки State Machine on Steroids удалось устранить указанные недостатки библиотеки Acts as State Machine:

— при наследовании сохраняется иерархия родительского автомата. Это позволяет обратиться к ранее созданным классам, например, к классу AdvancedUser::Deleted, а также решить задачу автоматического изоморфного переноса кода в модель;

— реализован метод для создания групп состояний, что сокращает необходимое число переходов при описании автомата в исполняемом коде.

Сравнение библиотек Acts as State Machine и State Machine on Steroids

Библиотека State Machine on Steroids сохраняет все достоинства динамических и объектно-ориентированных свойств языка Ruby, такие как:

- возможность создания кода в терминах предметной области. В результате эксперты могут понимать, проверять и модифицировать написанный код;
- обеспечение возможности разработки самодокументированного кода;
- повышение качества, надежности и сопровождаемости программ;
- сохранение иерархии автоматов при наследовании.

Еще одним достоинством библиотеки State Machine on Steroids является возможность изоморфного переноса графической нотации в код. При этом, в частности, удается исключить дублирование кода, которое возникает при использовании библиотеки Acts as State Machine при реализации групповых переходов.

В таблице показано необходимое число переходов и состояний для реализации автомата, представленного на рис. 3, с использованием указанных выше библиотек. Отметим, что библиотека State Machine on Steroids за счет исключения избыточности упрощает код.

■ *Сравнение эффективности библиотек*

Библиотека	Переходы	Состояния	Группы
Acts as State Machine	20	8	–
State Machine on Steroids	9	8	3

Заключение

В настоящей работе были рассмотрены две библиотеки. Их сравнение выполнено на примере модификации функциональности регистрации пользователей в плагине Restful-authentication.

Библиотеки позволяют переносить графическую нотацию в программный код. При этом разработанная библиотека State Machine on Steroids сохраняет иерархию родительского автомата при наследовании и обеспечивает изоморфную реализацию вложенных групп состояний.

Кроме того, при использовании ее возможен реверсинжениринг — автоматический изоморфный перенос кода в графическую модель.

Литература

1. **TIOBE.** TIOBE Software: Tiobe Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
2. **Тимофеев К. И., Астафуров А. А., Шалыто А. А.** Наследование автоматных классов с использованием динамических языков программирования на примере Ruby: Software Engineering Conference. Russia, 2008. http://www.secr.ru/?pageid=4548&submissionid=5270
3. **David H. H.** Ruby on Rails. http://www.rubyonrails.com/
4. **Grant G.** Restful Authentication Generator. http://github.com/technoweenie/restful-authentication/tree/master
5. **Szinek P.** Rails Rumble Observations. P. II: Trends in gem/plugin usage. http://www.rubyrailways.com/rails-rumble-observations-part-ii-trends-in-gemplugin-usage/
6. **Scott B.** Acts as State Machine. http://agilewebdevelopment.com/plugins/acts_as_state_machine.
7. **Object Management Group.** Official UML Specification. http://www.uml.org/#UML2.0
8. **Шопырин Д. Г., Шалыто А. А.** Графическая нотация наследования автоматных классов // Программирование. 2007. № 5. С. 62–74.