Опубликовано в Научно-техническом вестнике СПбГУ ИТМО. 2008. Выпуск 53. Автоматное программирование, с. 258-263.

УДК 004.432.4

ТЕКСТОВЫЙ ЯЗЫК АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

В.С. Гуров, М.А. Мазин, А.А. Шалыто

(Санкт-Петербургский государственный университет информационных технологий, механики и оптики)

В настоящей работе описывается текстовый язык автоматного программирования, построенный на основе системы метапрограммирования JetBrains MetaProgramming System (MPS). Этот язык лишен таких недостатков графического языка автоматного программирования, как низкая скорость ввода диаграмм, и более привычен для программиста.

Ключевые слова: автоматное программирование, текстовый язык

В рамках проекта *UniMod* [1] предложены метод и средство для моделирования и реализации объектно-ориентированных программ со сложным поведением на основе автоматного подхода. Модель системы в рамках указанного проекта предлагается строить с помощью двух типов *UML*-диаграмм: диаграммы классов и диаграммы состояний [2]. При этом на диаграмме классов, которая представляется в виде схемы связей и взаимодействия автоматов, изображаются источники событий, автоматы и объекты управления, которые реализуют функции входных и выходных воздействий. Код для источников событий и объектов управления пишется на языке *Java* [3]. Поведение автоматов описывается с помощью диаграмм состояний.

При помощи инструментального средства *UniMod* выполнен ряд проектов, которые доступны по адресу http://is.ifmo.ru/unimod-projects/. Данные проекты показали эффективность применения автоматного программирования и средства *UniMod* при реализации систем со сложным поведением, но также выявили и ряд недостатков:

- ввод диаграмм состояний с помощью графического редактора трудоемок;
- многие программисты предпочитают работать с текстовым представлением программы, несмотря на то, что диаграммы позволяют представлять информацию более компактно и обозримо;
- невозможно в одном *Java*-классе совместить автомат и объект управления, что не позволяет прозрачно использовать автоматное пограммирование совместно с объектно-ориентированным, так как в настоящее время код, генерируемый из автоматной модели, не является в полной мере объектно-ориентированным.

В настоящий момент существует несколько текстовых языков, поддерживающих программирование с применением автоматов, как в рамках парадигмы автоматного программирования (UniMod FSML [4], State Machine [5]), так и вне ее (SMC [6], TABP [7], SCXML [8], Ragel [9]). Каждый из этих языков программирования обладает некоторыми недостатками и ограничениями.

Язык State Machine представляет собой расширение языка Java, основанное на паттерне проектирования State Machine [10]. Этот паттерн позволяет в объектно-ориентированном стиле описывать структуру автомата. Однако такое описание оказывается громоздким, что препятствует использованию языка State Machine в реальной разработке.

Язык SMC – универсальный компилятор автоматов. Этот язык позволяет описывать в автоматы в едином формате и генерировать целевой код на различных языках общего назначения. В то же время интеграция этого языка с целевыми языками требует от разработчиков дополнительных усилий – необходимо существенно модифицировать код класса для того, чтобы связать его с автоматом, описывающим его поведение.

В универсальном языке программирования TABP предусмотрены встроенные конструкции, облегчающие написание автоматов. Однако синтаксис этого языка вы-

нуждает разработчика оперировать понятиями более низкоуровневыми, чем «состояние». Кроме того, универсальность языка ТАВР доставляет дополнительные трудности разработчику, связанные с необходимостью изучать конструкции нового языка даже для тех задач, для которых подходят более распространенные универсальные языки.

Язык Ragel предназначен для описания конечных автоматы с помощью регулярных выражений. Такой подход ограничивает область применимости этого языка задачами лексического анализа и спецификации протоколов,

Стандарт языка SCXML – спецификация XML-представления автоматов Харела [11]. Использовать этот язык непосредственно для написания кода автоматов на нем крайне затруднительно, скорее он подходит для обмена автоматами Харела между приложениями.

Для устранения перечисленных недостатков авторы предлагают новый подход к разработке автоматных программ и применению автоматов в объектно-ориентированных системах. В рамках этого подхода предлагается использовать систему метапрограммирования JetBrains Meta Programming System (MPS) [12, 13], которая позволяет создавать проблемно-ориентированные языки (Domain Specific Language – DSL) [13, 14].

Для задания языка в системе MPS требуется разработать:

- структуру абстрактного синтаксического дерева (АСД) [15] для разрабатываемого языка. Узлам АСД могут соответствовать такие понятия, как «объявление класса», «вызов метода», «операция сложения» и т.п.;
- модель текстового редактора для каждого типа узла АСД. Задание редактора для узла АСД равноценно заданию конкретного синтаксиса для этого узла. При этом если для традиционных текстовых языков программирования создание удобного редактора – отдельная сложная задача, то для языков, созданных с помощью средства MPS, редакторы являются частью языка. Эти редакторы поддерживают автоматическое завершение ввода текста и проверку корректности программы;
- модель ограничений на экземпляры АСД;
- модель системы типов [16] для языка;
- модель трансформации программы на задаваемом языке в исполняемый код.

Система *MPS* позволяет как создавать новые языки, так и расширять языки, уже созданные с помощью этой системы.

В отличие от традиционных языков, языки, созданные с помощью системы *MPS*, не являются текстовыми в общепринятом смысле, так как при программировании на них пользователь пишет не текст программы, а вводит ее в виде АСД с помощью специальных редакторов. Структура и внешний вид этих редакторов таковы, что работа с моделью программы для пользователя выглядит, как традиционная работа с текстом программы. Отказ от традиционного текстового ввода программ значительно упрощает создание новых языков [17] – исчезает необходимость в разработке лексических и синтаксических анализаторов, и, как следствие, перестают действовать ограничения на класс грамматик языков. Недостатком такого подхода является зависимость языков от системы *MPS* – невозможно разрабатывать программы без этой системы. Однако подобное ограничение присуще и традиционным, чисто текстовым языкам, которые зависят от компиляторов. Впрочем, после трансляции программы, написанной на языке, созданном в системе *MPS*, исполняемый код перестает зависеть от этой системы.

Ядро среды *MPS* написано на кросс-платформенном языке *Java*. В связи с этим в среде *MPS* существуют развитые средства для взаимодействия с *Java*-платформой. Для написания *Java*-кода в среде *MPS* разработан язык *baseLanguage*. Этот язык является почти полной реализацией спецификации *Java* 5 [18]. В нем определены такие конструкции, как «класс», «интерфейс», «метод», «предложение», «выражение» и т.д.

Язык для автоматного программирования в среде *MPS* получил название *stateMachine*. Он представляет собой автоматное расширение языка *baseLanguage*. Основной целью, которая преследовалась при разработке языка *stateMachine*, было создание средства, позволяющего описывать поведение классов в виде автоматов, не накладывая на сами классы никаких дополнительных ограничений. Более того, автоматное описание поведения класса должно быть инкапсулировано. Это означает, что код, использующий класс, не знает о том, каким образом задано поведение класса. Этот подход отличается от предложенного в работе [1] тем, что позволяет использовать автоматы не только в программах, написанных в соответствии со SWITCH-технологией, но и в традиционных объектно-ориентированных программах.

В качестве примера использования языка *stateMachine* рассмотрим систему управления лифтом [19]. Используемая реализация системы является упрощенным решением задачи управления лифтом, описанной в работе [20]. Логика управления всеми подсистемами лифта реализована в одном автомате.

Каждый автомат в языке *stateMachine* связан с некоторым классом и описывает его поведение. Чтобы задать поведение класса с помощью автомата, необходимо в этом классе определить события, на которые будет реагировать автомат. Особенностью языка *stateMachine* является то, что события в нем – это методы специального вида. В языке *Java* декларации методов различаются по способу реализации:

- обычные методы, реализация которых следует сразу за объявлением метода;
- нативные методы, реализованные на платформо-зависимых языках программирования;
- абстрактные методы, вообще не имеющие реализации.

```
public class Elevator extends <none>  implements DoorsEngineListener    {
                                                  ElevatorEngineListener
                                                  LoadingTimerListener

  <<initializer>>
  <<static fields>>
  public int currentFloor = 1;
  public TaskList tasks = new TaskList();
  public ElevatorEngine elevatorEngine;
  public DoorsEngine doorsEngine;
  public LoadingTimer loadingTimer;
  <<properties>>

  public Elevator(ElevatorEngine elevatorEngine, DoorsEngine doorsEngine,
    this.elevatorEngine = elevatorEngine;
    this.doorsEngine = doorsEngine;
    this.loadingTimer = loadingTimer;
    this.elevatorEngine.addElevatorEngineListener(this );
    this.doorsEngine.addDoorsEngineListener(this );
    this.loadingTimer.addLoadingTimerListener(this );
  }


  public event doorsOpened( );
  public event doorsClosed( );
  public event floorReached(int floor);
  public event call(int floor, CallType callType);
  public event openDoors( );
  public event loadingTimeout( );
  public event fire( );
```

Рисунок. События автомата, управляющего лифтом

Событие в языке *stateMachine* – это метод, реализация которого находится в автомате и зависит от текущего состояния автомата. Например, в классе *Elevator* объявлены следующие события (см. рисунок):

- «*doorsOpened*», «*doorsClosed*» – события, которые посылает объект управления *DoorsEngine,* когда двери оказываются в максимально открытом и полностью закрытом положении соответственно;
- «*floorReached*» – событие, которое посылает объект управления *ElevatorEngine,* когда лифт достигает очередного этажа;
- «*call*» – событие, которое получает автомат, когда пассажир нажимает кнопку вызова лифта на этаже, или в кабине лифта;
- «*openDoors*» – событие, которое получает автомат, когда пассажир нажимает кнопку экстренного открывания дверей в кабине лифта;
- «*loadingTimeout*» – событие, которое посылает объект управления *LoadingTimer,* когда истекает время ожидания погрузки пассажиров.

Так как события являются обычными методами, их можно использовать в качестве реализации абстрактных методов. Это позволяет уменьшить количество зависимостей в программе. Например, объект управления *ElevatorEngine* не имеет непосредственных ссылок на класс *Elevator*. Вместо этого в программе управления лифтом объявлен интерфейс *ElevatorEngineListener*, и объект управления *ElevatorEngine* извещает о событиях экземпляры этого интерфейса. Класс *Elevator* реализует интерфейс *ElevatorEngineListener* и поэтому может обрабатывать события объекта управления *ElevatorEngine*. Таким образом, применяется классический паттерн проектирования «Обозреватель» [21].

Реакция автомата на событие зависит от состояния, в котором автомат находится. Состояния автомата бывают двух типов – обычные и конечные. При этом конечные состояния не имеют исходящих переходов. Таким образом, когда автомат оказывается в конечном состоянии, он перестает обрабатывать события.

Первое по порядку состояние, объявленное в автомате, считается начальным. При создании экземпляра класса, для которого определен автомат, после выполнения конструктора осуществляется переход в начальное состояние.

Обычные состояния могут быть вложены друг в друга. При этом если состояние содержит другие состояния, то оно называется составным. При переходе в составное состояние выполняется переход в первое по порядку вложенное в него состояние. Поэтому автомат после обработки события не может оказаться в составном состоянии. Каждый исходящий из составного состояния переход работает так, как если бы такой переход был добавлен к каждому вложенному состоянию.

Каждый автомат в языке *stateMachine* связан с некоторым классом, имеет доступ к его полям и методам. Поэтому в языке *stateMachine* нет необходимости в специальных конструкциях для взаимодействия между автоматами, так как вместо вложения одного автомата в другой можно использовать агрегацию одного класса с автоматом, другим классом с автоматом, а посылка события из одного автомата в другой есть не что иное, как вызов метода. После написания программы на языке *stateMachine* она сначала транслируется в *Java*-код, а затем компилируется стандартным *Java*-компилятором. Преимуществом этого языка является простота его использования в объектно-ориентированных приложениях, написанных на языке *Java*. При применении этого языка проверка корректности программы осуществляется на стадии ее написания, а не в процессе компиляции.

Исходя из изложенного, можно утверждать, что в настоящей работе предложен подход к построению программ, который качественно упрощает применение автоматов в объектно-ориентированных программах.

Литература

1. Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А. UML. SWITCH-Технология. Eclipse // Информационно-управляющие системы. – 2005. – №6(13). – С. 12–17. – Режим доступа: http://is.ifmo.ru/works/uml-switch-eclipse/
2. Гуров В.С., Мазин М.А., Шалыто А.А. Операционная семантика UML-диаграмм состояний в программном пакете UniMod / Труды XII Всероссийской научно-методической конференции "Телематика-2005". – СПб: СПбГУ ИТМО, 2005. – Т.1. – С.74–76. – Режим доступа: http://is.ifmo.ru/works/uml-switch-eclipse/
3. Гуров В., Мазин М., Нарвский А., Шалыто А. UML. SWITCH-технология. Eclipse// «Информационно-управляющие системы», 2004, № 6, с.12-17. – Режим доступа: http://is.ifmo.ru/works/uml-switch-eclipse/
4. Гуров В.С., Мазин М.А., Шалыто А.А. Операционная семантика UML-диаграмм состояний в программном пакете UniMOD. – Режим доступа: http://tm.ifmo.ru/tm2005/db/doc/get_thes.php?id=224
5. Eckel B. Thinking in Java. – NJ: Prentice Hall, 2006.
6. Лагунов И.А. Разработка текстового языка автоматного программирования и его реализация для инструментального средства UniMod на основе автоматного подхода. – 2008.
7. Шамгунов Н.Н., Корнеев Г.А., Шалыто А.А. State Machine — расширение языка Java для эффективной реализации автоматов // Информационно-управляющие системы. – 2005. – № 1. – С. 16–24.
8. SMC – State Machine Compiler. – Режим доступа: http://smc.sourceforge.net/.
9. Цымбалюк Е. А. Текстовый язык автоматного программирования ТАВР. – 2008.
10. State Chart XML (SCXML): State Machine Notation for Control Abstraction. W3C Working Draft 16 May 2008. – Режим доступа: http://www.w3.org/TR/2008/WD-scxml-20080516/
11. Thurston A. Parsing Computer Languages with an Automaton Compiled from a Single Regular Expression. / 11th International Conference on Implementation and Application of Automata (CIAA 2006), Lecture Notes in Computer Science, volume 4094, pp. 285—286. – Taipei, Taiwan, August 2006.
12. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Reading, Mass., 1995.
13. Harel D. Statecharts: A visual formalism for complex systems //Science of Computer Programming. – 1987. - № 8, pp. 231–274.
14. Дмитриев С. Языково-ориентированное программирование: следующая парадигма //RSDN Magazine. – 2005. – № 5.
15. Фаулер М. Языковой инструментарий: новая жизнь языков предметной области – Режим доступа: http://www.maxkir.com/sd/languageWorkbenches.html
16. Ward M. Language Oriented Programming //Software — Concepts and Tools, 15, 1994.
17. Ахо А., Сети Р., Ульман Дж. Компиляторы. Принципы, технологии, инструменты. – М.: Вильямс, 2003.
18. Luo Z. Computation and Reasoning: A Type Theory for Computer Science. – Oxford University Press, 1994.
19. Simonyi C. The Death of Computer Languages, the Birth of Intentional Programming // The Future of Software, Univ. of Newcastle upon Tyne, England, Dept. of Computing Science, 1995.
20. J. Gosling, B. Joy, G. Steele, G. Bracha. The Java Language Specification, Third Edition – Addison Wesley, 2005.
21. Кнут Д. Искусство программирования. Т. 1: Основные алгоритмы. – М.: Вильямс, 2001. – 712 с.

22. Наумов Л. А., Шалыто А. А. Искусство программирования лифта. Объектно-ориентированное программирование с явным выделением состояний // Информационно-управляющие системы. – 2003. – № 6. – С.38–49; Мир ПК – Диск. – 2004. – № 2. – С 18. – Режим доступа: http://is.ifmo.ru.
23. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison Wesley, Reading, Mass., 1995.