СТАТЬЯ ОПУБЛИКОВАНА В «НАУЧНО-ТЕХНИЧЕСКОМ ВЕСТНИКЕ СПбГУИТМО». ВЫП. 42 ФУНДАМЕНТАЛЬНЫЕ И ПРИКЛАДНЫЕ ИССЛЕДОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ. 2007, с. 29-32

ТЕКСТОВЫЙ ЯЗЫК АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

В. С. Гуров, М. А. Мазин, А. А. Шалыто

В статье описывается текстовый язык автоматного программирования, созданный с помощью системы метапрограммирования *JetBrains MPS*. Этот язык дополняет визуальное инструментальное средство автоматного программирования *UniMod* текстовым входом. При этом диаграммы состояний строятся по мере набора текста программы автоматически, что позволяет назвать предлагаемый подход к построению программ текстово-визуальным.

В рамках проекта *UniMod* [1] предложены метод и средство для визуального моделирования и реализации объектно-ориентированных программ со сложным поведением на основе автоматного подхода. При этом системы предлагалось строить с помощью двух типов *UML*-диаграмм: диаграмм классов и диаграмм состояний [2]. 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. Поведение автоматов описывается с помощью диаграмм состояний. Код для источников событий и объектов управления пишется на языке *Java* [3]. Указанные диаграммы и написанный вручную код могут интерпретироваться или компилироваться. Поэтому проекта *UniMod* указанные диаграммы являются не картинками, а визуальным языком программирования. Таким образом, этот проект поддерживает концепцию *Исполняемый UML* (*Executable UML*) [4, 5].

С использованием инструментального средства *UniMod* выполнен ряд проектов, который доступны по адресу http://is.ifmo.ru/unimod-projects/. Эти 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.

Для устранения перечисленных недостатков авторы предложили новый подход к разработке автоматных программ и применению автоматов в объектно-ориентированных системах.

В рамках этого подхода используется система метапрограммирования *JetBrains Meta Programming System (MPS)* [6-8], которая позволяет создавать проблемно-ориентированные языки (*Domain Specific Language — DSL*) [8, 9]. Для задания языка в системе *MPS* требуется разработать:

- структуру абстрактного синтаксического дерева (АСД) [10] 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 [11] 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 традиционном понимании, 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 [12] – 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.

С помощью *MPS* авторами созданы два варианта текстового языка для автоматного программирования.

Первый язык выполнен в виде самостоятельного языка, а второй – в виде расширения языка *Java*. Эти языки позволяют описывать состояния и логику переходов автоматов, а также события, обрабатываемые автоматами. При этом, также как и в инструментальном средстве *UniMod*, функции входных и выходных переменных реализуются на другом языке программирования.

На рис. 1 показана структура абстрактного синтаксического дерева текстового языка автоматного программирования первого типа.

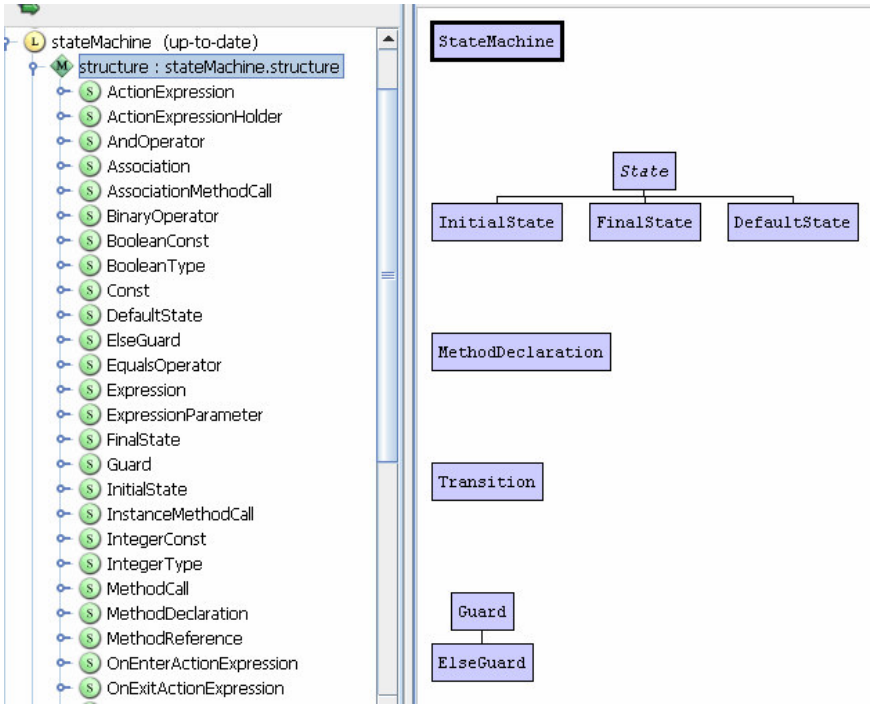

Рис. 1. Структура абстрактного синтаксического дерева текстового языка автоматного программирования первого типа в среде *MPS*

Текстовый язык автоматного программирования второго типа расширяет язык *Java* и позволяет в одном *Java*-классе совмещать автоматные и не автоматные аспекты. После написания программы на этом языке, она сначала транслируется в *Java*-код, но уже без автоматного расширения, 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.

В то же время для приложений, написанных на других языках, Этот язык не подходит. Для таких приложений препочтительно использовать независимую от языка *Java*, первую версию автоматного языка. Эта версия менее зависима от платформы и полностью соответствует понятию запускаемых спецификаций.

Использование специальных языков для автоматного программирования упрощает разработку автоматов, избавляя программиста от необходимости описывать их средствами языков общего назначения.

Так как диаграммы состояний, являющиеся более наглядными по сравнению с текстовым представлением автоматов, то при использовании рассматриваемых языков может быть обеспечена возможность автоматического построения диаграммы состояний по мере набора текста. Однако эти диаграмму доступны только в режиме просмотра, а не редактирования.

На рис. 2 слева показан пример программы на разработанном текстовом языке автоматного программирования первого типа, который распознает цепочки символов вида *a*b*c**. Автоматически построенная диаграмма состояний изображена на рис. 2 справа.

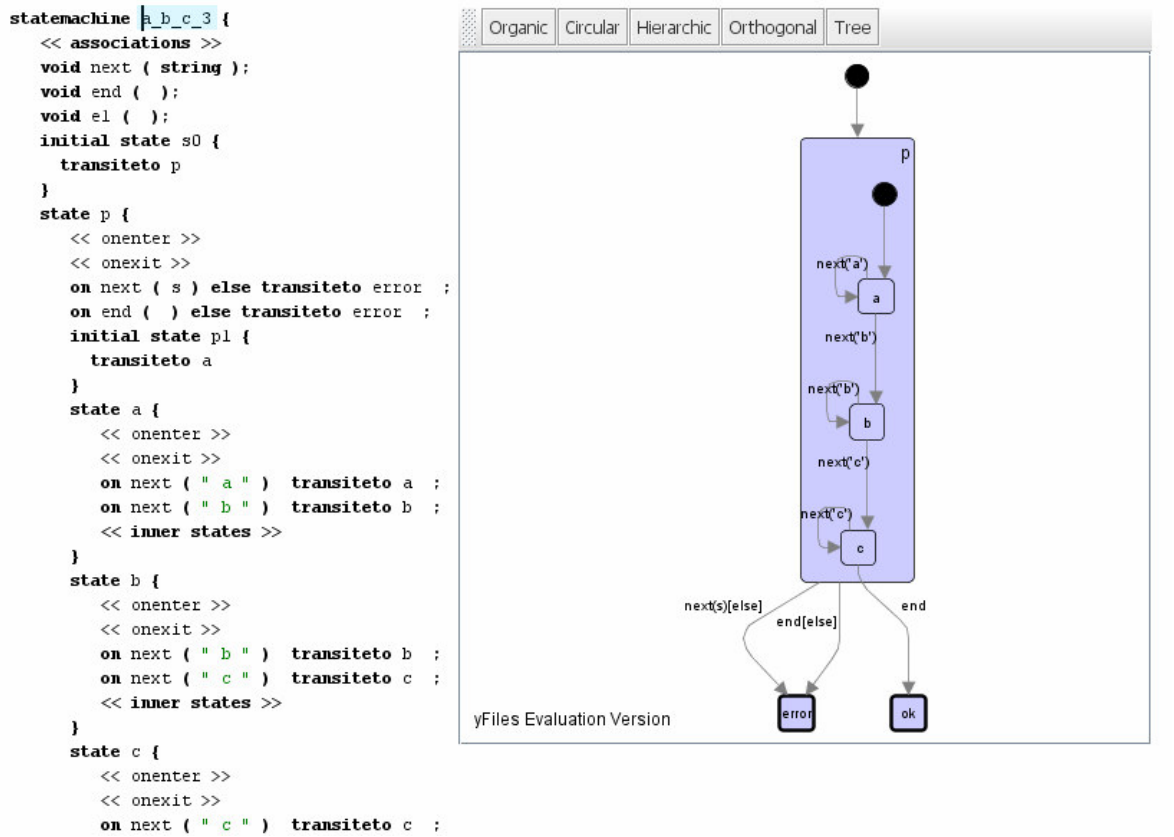

Рис. 2. Пример программы на текстовом языке автоматного программирования и автоматически построенной по ней диаграммы состояний

С применением текстового языка автоматного программирования второго типа можно подробно ознакомиться в работе [13].

Исходя из изложенного, можно утверждать, что в настоящей работе предложен подход к построению программ, который может быть назван текстово-визуальным.

Литература

1. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-Технология. Eclipse //Информационно-управляющие системы. 2005. № 6, с. 12–17. 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". СПбГУ ИТМО. Т.1, с.74–76. http://tm.ifmo.ru/tm2005/db/doc/get_thes.php?id=224
3. *Eckel B.* Thinking in Java. NJ: Prentice Hall, 2006.
4. *Executable UML.* http://en.wikipedia.org/wiki/Executable_UML
5. *Гуров В.С., Нарвский А.С., Шалыто А.А.* Исполняемый UML из России // PC Week/RE. 2005. № 26, с. 18, 19. http://is.ifmo.ru/works/_umlrus.pdf
6. *Дмитриев С.* Языково-ориентированное программирование: следующая парадигма //RSDN Magazine. 2005. № 5. http://www.rsdn.ru/article/philosophy/LOP.xml

7. *Fowler M.* A Language Workbench in Action – MPS http://martinfowler.com/articles/mpsAgree.html
8. *Фаулер М.* Языковой инструментарий: новая жизнь языков предметной области http://www.maxkir.com/sd/languageWorkbenches.html
9. *Ward M.* Language Oriented Programming //Software – Concepts and Tools. 1994. № 15.
10. *Ахо А., Сети Р., Ульман Дж.* Компиляторы. Принципы, технологии, инструменты. М.: Вильямс, 2003.
11. *Luo Z.* Computation and Reasoning: A Type Theory for Computer Science. Oxford University Press, 1994.
12. *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.
13. *Красильников Н. Н., Шалыто А. А.* Мультиагентная система дорожного движения. Реализация на языке Java и текстовом языке автоматного программирования. СПбГУ ИТМО. 2007. http://is.ifmo.ru/download/vihicles/doc/vihicles2.pdf