УДК 004.434

АВТОМАТНЫЙ МЕТОД ОПРЕДЕЛЕНИЯ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ (Часть 3)[1]

Ф. А. Новиков,
канд. физ.-мат. наук, заведующий лабораторией
Институт прикладной астрономии РАН
У. Н. Тихонова,
аспирант
Санкт-Петербургский государственный политехнический университет

Описывается новый метод определения синтаксиса и семантики проблемно-ориентированных языков с помощью диаграмм классов и диаграмм автоматов. В третьей части статьи описывается задание операционной семантики системами взаимодействующих автоматов на примере мини-языка множеств.

Ключевые слова *— проблемно-ориентированный язык, абстрактный синтаксис, метамодель, автоматное программирование.*

Определение операционной семантики системой интерпретирующих автоматов

Рассмотрев в предыдущих частях статьи определения метамодели и конкретного синтаксиса проблемно-ориентированного языка с помощью системы взаимодействующих автоматов, перейдем к самому проблематичному вопросу — к семантике.

Чтобы описать операционную семантику, необходимо, во-первых, определить модель вычислимости, т. е. виртуальную машину, и, во-вторых, задать либо преобразование абстрактной программы в программу виртуальной машины, либо интерпретацию абстрактной программы виртуальной машиной. При этом для одного и того же языка можно предложить различные с прагматической точки зрения семантики. То есть возможны совершенно разные способы использования одного и того же проблемно-ориентированного языка. Приведем несколько характерных примеров:

- преобразование входа программы в ее выход — системы пакетной обработки;
- последовательность побочных эффектов в процессе выполнения программы — командные, интерактивные системы управления;
- сервис или служба, отвечающая на запросы пользователя, — интеллектуальные экспертные системы.

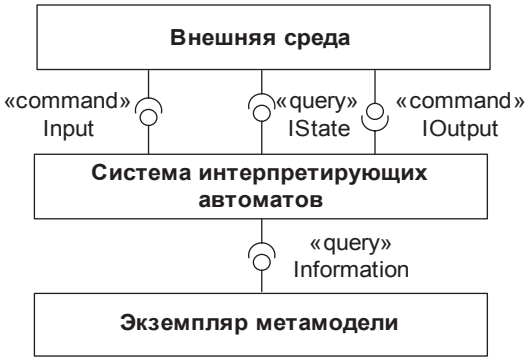
Например, для мини-языка множеств в первом случае результатом работы программы является набор значений построенных множеств. Наиболее очевидным способом реализации такой семантики является использование булевой матрицы M [Имя, Буква]: Boolean в качестве результата работы программы. В этой матрице M [name, letter] = true означает, что множество с именем name содержит элемент с буквой letter[2]. Тогда алгоритм интерпретации абстрактной программы (экземпляра метамодели), в частности алгоритм вычисления выражений над множествами, может быть запрограммирован известным образом [1]. Описание этого алгоритма с помощью автоматов не дает особых преимуществ, и мы на нем не останавливаемся.

Во втором случае интерес представляет протокол выполнения программы, т. е. последовательность побочных эффектов, связанных с появлением и удалением элементов в множествах. Такую семантику мини-языка множеств можно придумать с некоторой натяжкой, поэтому ее мы также не рассматриваем.

В третьем случае программу можно рассматривать как систему уравнений, задающую предметную область, например $A = B \cap \{b\}$. Поставив такой программе вопрос: «A = ? при условии B = {...}», пользователь получает в ответ значение множества A. Если при этом требуется реализовать бо-

[1] Окончание. Начало в № 6, 2009; № 2, 2010.

[2] Видимо, это простейшее возможное в данном случае представление данных.

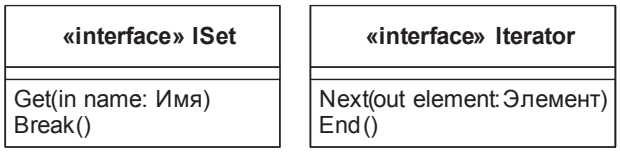


■ ***Рис. 24.*** *Схема взаимодействия автоматов семантики*

лее сложную прагматику, например, не просто вычисление значения множества, а итератор, перебирающий элементы множества, то применение автоматного метода оказывается вполне оправданным и естественным.

В автоматном методе семантика задается системой взаимодействующих автоматов, интерпретирующих экземпляр метамодели (см. описание автоматной модели в разделе «Автоматный метод и модель системы автоматов» [2]). В общем случае система автоматов взаимодействует, с одной стороны, с экземпляром метамодели (абстрактной программой), черпая оттуда информацию для интерпретации, с другой стороны — с некоторой *внешней средой*, получая от нее запросы и команды и выдавая ответы (рис. 24).

Рассмотрим пример задания семантики мини-языка множеств в следующей постановке. Имеется экземпляр метамодели, заданной на рис. 12 [2]. Пользователь (элемент внешней среды) задает имя множества и получает в ответ последовательность событий — элементов этого множества или ничего не получает, если такое множество не определено.



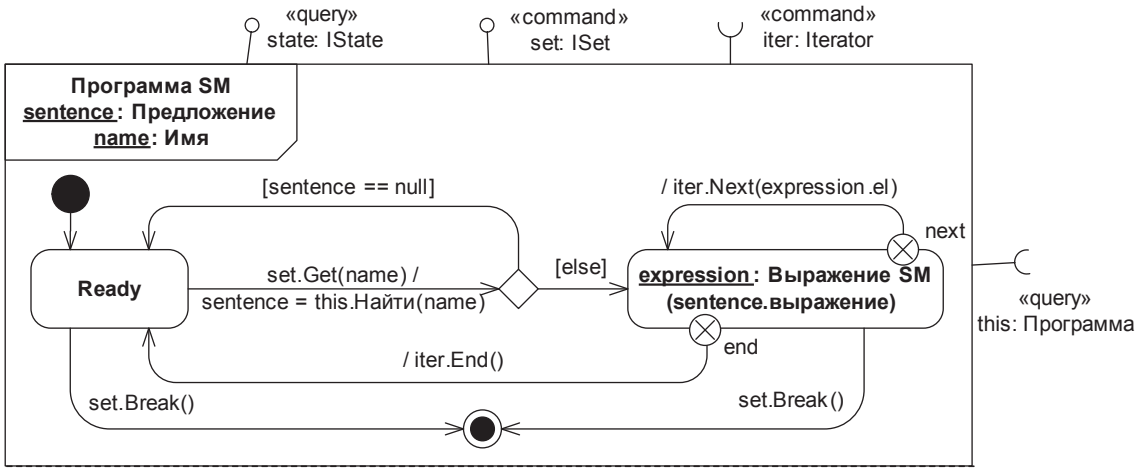
■ ***Рис. 25.*** *Интерфейсы взаимодействия автоматов семантики мини-языка множеств*

Для реализации такой семантики определим следующие интерфейсы взаимодействия: интерфейс ISet позволяет задать имя множества, а интерфейс Iterator воплощает итератор (рис. 25).

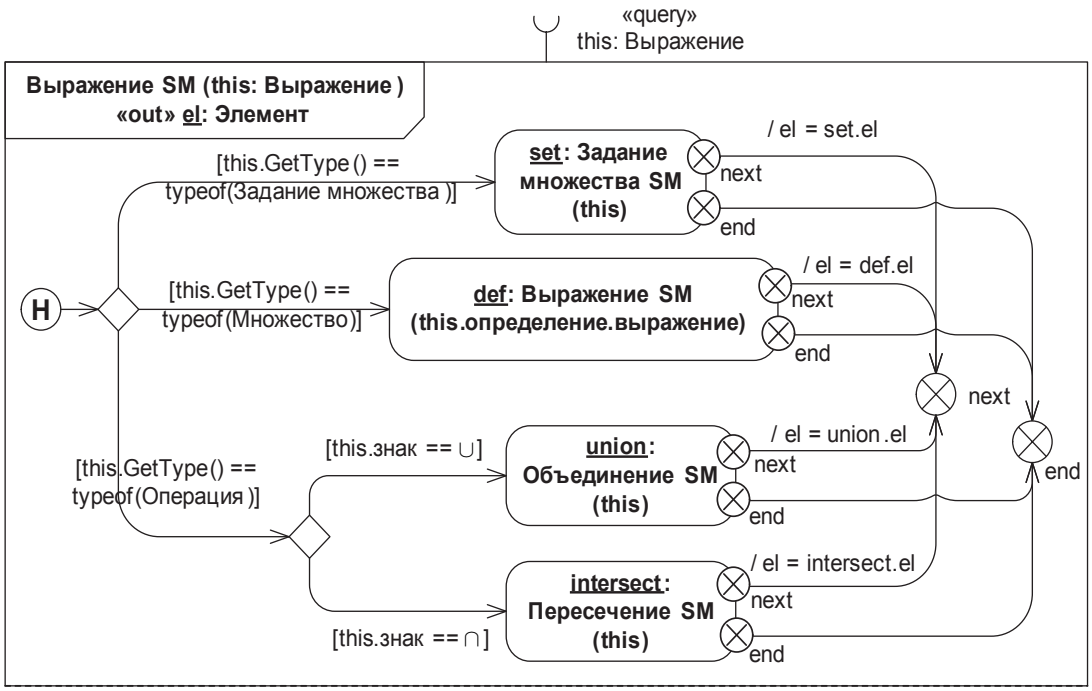
В таком случае головной автомат, соответствующий классу Программа, предоставляет интерфейс ISet и требует интерфейс Iterator, через который события (элементы итерируемого множества) отдаются внешней среде, и интерфейс this: Программа, через который обеспечивается взаимодействие с экземпляром класса Программа в абстрактной программе. Через интерфейс IState внешняя среда может контролировать выполнение процесса (рис. 26).

В этом автомате состояние Ready соответствует готовности автомата выполнить работу. Второе состояние является составным, в него вложен автомат интерпретации выражения. Заметим, что экземпляру автомата Выражение SM передается в качестве параметра объект класса Выражение, который этот автомат должен интерпретировать. Тем самым задается связь с метамоделью. Для получения данных (результирующих элементов итерируемого множества) из вложенного автомата используется локальная переменная el автомата expression: Выражение SM.

В автомате, который выбирает способ интерпретации выражения в зависимости от его типа (рис. 27), мы использовали нестандартный стереотип «out» для указания того, что локальная пере-



■ ***Рис. 26.*** *Головной автомат семантики мини-языка множеств*



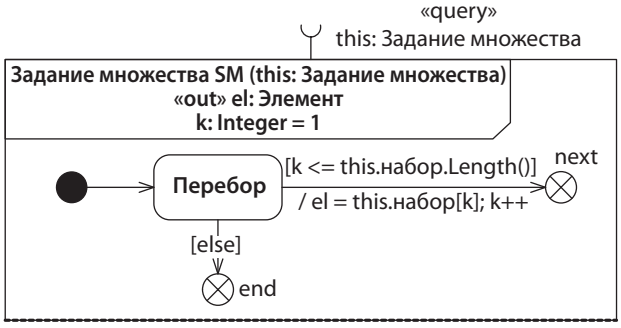
■ ***Рис. 27.*** *Автомат интерпретации выражения мини-языка множеств*

менная является результатом работы автомата, который можно получить извне.

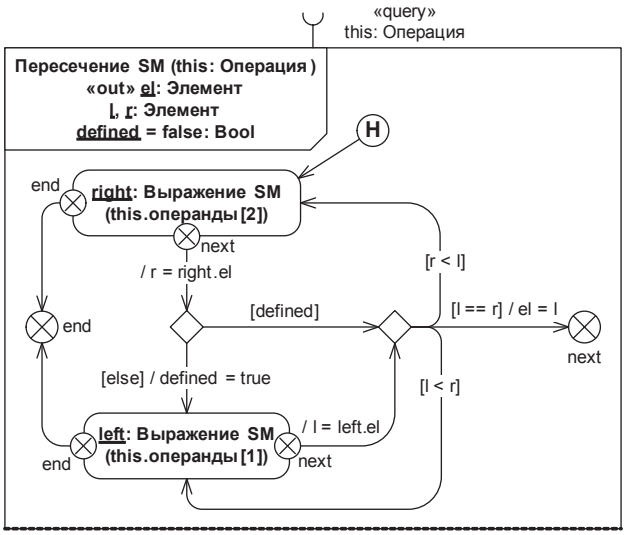
Как этот, так и следующие автоматы (рис. 28–30) за один проход от начального до конечного состояния (в качестве которого здесь используются точки выхода) выдают очередной элемент множества и сохраняют его в своей переменной el. Непосредственно перебор всех элементов множества осуществляется в головном автомате петлей у состояния expression (см. рис. 26).

Автомат для самого простого случая выражения в мини-языке множеств — задания множества своими элементами — приведен на рис. 28.

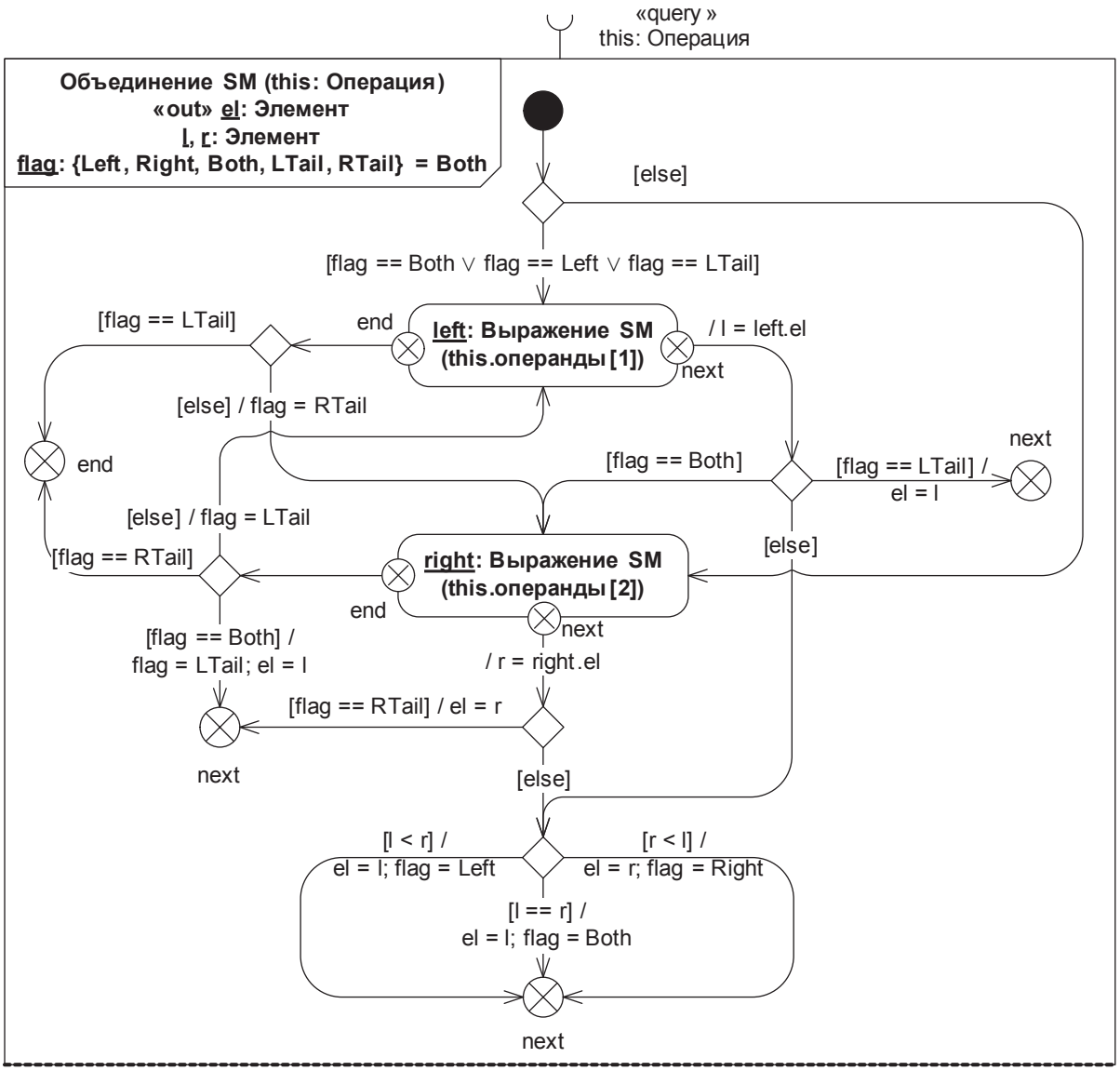
Автомат, интерпретирующий операцию пересечения множеств путем *слияния* упорядоченных множеств, представлен на рис. 29. Алгоритм параллельно просматривает два множества, причем на каждом шаге продвижение происходит в том множестве, в котором текущий элемент меньше [3]. Здесь мы опираемся на допущение, что элементы множеств упорядочены в алфавитном порядке и операндов два. Чтобы отследить, что обе переменные l и r (в которых хранятся очередные элементы множеств левого и правого операндов соответственно) были проинициализированы, используется флаг defined. Инициализация локальной переменной при ее объявлении (как это сделано для переменной defined) означает, что переменная получает указанное значение сразу же после создания экземпляра автомата.



■ ***Рис. 28.*** *Автомат интерпретации задания множества в мини-языке множеств*



■ ***Рис. 29.*** *Автомат интерпретации операции пересечения в мини-языке множеств*



■ ***Рис. 30.*** *Автомат интерпретации операции объединения в мини-языке множеств*

Наконец, на рис. 30 представлен автомат, интерпретирующий операцию объединения множеств путем того же алгоритма слияния упорядоченных множеств. Сложность этого автомата (сравн. рис. 29) определяется тем, что помимо его взаимодействия с автоматами, интерпретирующими левый и правый операнды (left и right соответственно), ему необходимо также хранить информацию о том, как был получен очередной элемент множества. От этого зависит, какую из переменных — l, r или обе вместе — нужно обновлять на следующем шаге. Для хранения этой информации используется переменная flag перечислимого типа. Кроме того, в отличие от пересечения множеств, когда исчерпание одного из операндов означает окончание вычисления, в операции объединения необходимо итерировать до конца оба операнда. В автомате на рис. 30 это делается с помощью значений перечислимого типа RTail и LTail.

Значения переменной flag логически соответствуют состояниям автомата, реализующего объединение множеств. Фактически было бы желательно, чтобы использовалась система состояний {Left, Right, Both, LTail, RTail}. Для этого необходимо реализовать взаимодействие автомата Объединение SM с автоматами операндов не с помощью вложенности, как во всех предыдущих примерах, а иным способом. Альтернативная модель взаимодействия между автоматами опирается на то, что каждый автомат может быть как источником событий, так и объектом управления для других автоматов (см. модель на рис. 10 [2]). В начале выполнения экземпляры автоматов создаются, свя-

зываются друг с другом через свои интерфейсы и затем работают параллельно.

Перейдем ко второму примеру описания той же самой семантики мини-языка множеств, в котором мы демонстрируем преимущества использования параллельно работающих автоматов[3].

Операционную семантику мини-языка множеств будут определять четыре класса автоматов. Модель взаимодействия между этими автоматами показана на рис. 31. Здесь классы *Выражение SM* и *Операция SM* введены только для удобства описания взаимодействия и не являются автоматами семантики. Автомат Программа SM позволяет выбрать множество по имени и передает внешней среде результаты его итерирования (элементы множества) в виде команд итератора. Эти элементы множества автомат Программа SM в свою очередь получает от итератора выражения, непосредственно определяющего множество. Автомат Задание множества SM реализует итератор множества перебором его элементов, хранящихся в метамодели. Автоматы Пересечение SM и Объединение SM реализуют итераторы соответствующих операций на основе алгоритмов слияния упорядоченных множеств, которые представлены итераторами операндов.

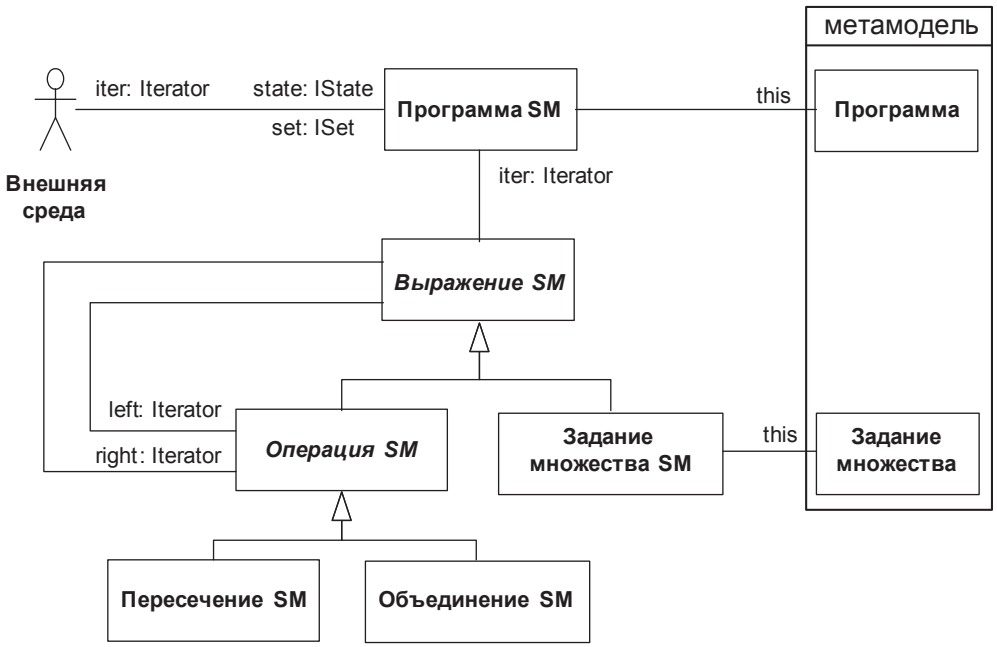
На основе экземпляра метамодели по определенным правилам трансформации моделей создаются и связываются друг с другом экземпляры автоматов. А именно, по экземпляру класса Программа (аксиома языка) создается единственный экземпляр автомата Программа SM. Остальные автоматы конструируются по экземплярам специализированных подклассов суперкласса Выражение. В данном случае правило трансформации удобно выразить с помощью следующего алгоритма:

```
proc CreateAuto (expr: Выражение, parent: Iterator) : Выражение SM
    var result: Выражение SM
    case expr.GetType() of
    typeof (Задание множества):
        result = new Задание множества SM()
        result.this = expr
        result.iter = parent
    typeof (Множество):
        result = CreateAuto(expr.определение.выражение, parent)
    typeof (Операция):
        if expr.знак == ∩ then
            result = new Пересечение SM()
        else    // expr.знак == ∪
            result = new Объединение SM()
        endif
        result.left  = CreateAuto (expr.операнды[1], result)
        result.right = CreateAuto (expr.операнды[2], result)
        result.iter  = parent
    end case
end proc
```

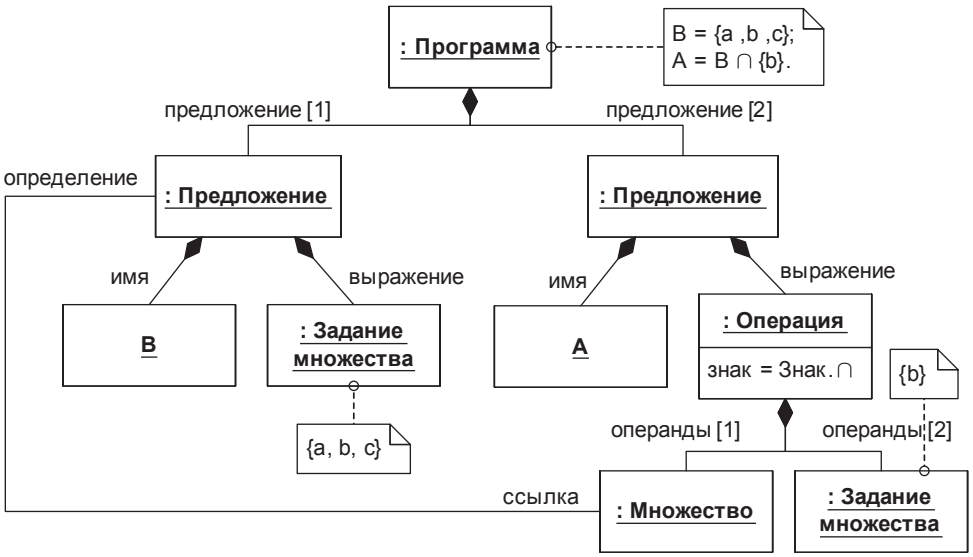
Таким образом, дерево взаимодействия автоматов, можно сказать, гомоморфно дереву абстрактной программы: сохраняется структура композиции и элиминируются ненужные при интерпретации перекрестные ссылки. Например, для абстрактной программы, представленной на рис. 32, соответствующая схема взаимосвязей автоматов показана на рис. 33.

На рис. 32 фактически приведен пример того, что называется *репозиторной моделью* при моде-

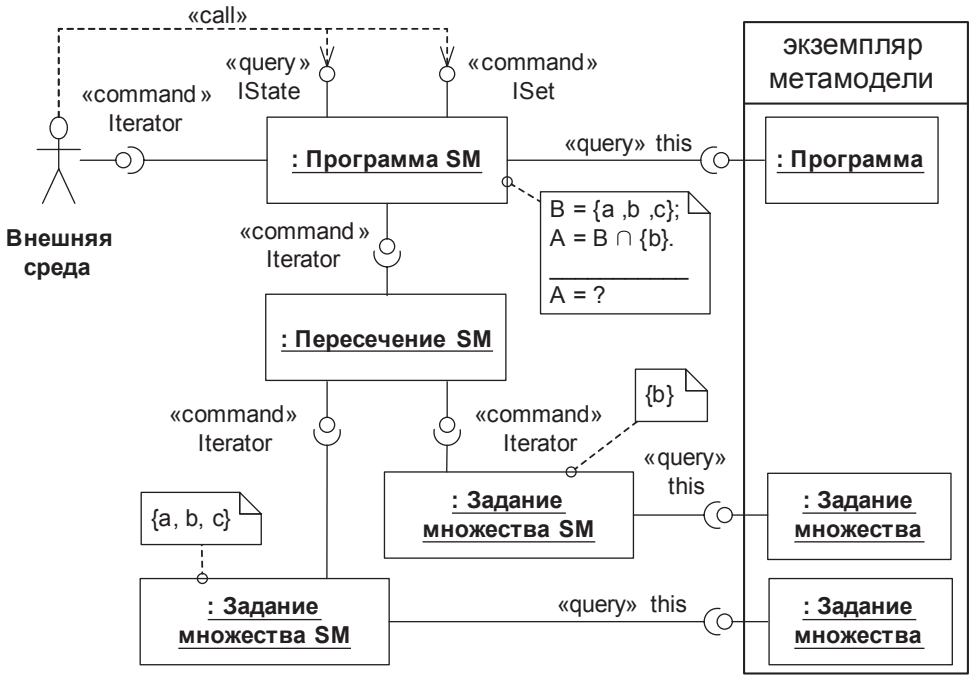
[3] Поскольку семантика та же самая, рисунки имеют сходные названия, и чтобы их отличить от предшествующих, мы добавляем к названиям рисунков пометку «(пример 2)».



■ ***Рис. 31.*** *Модель взаимодействия автоматов семантики (пример 2)*



■ ***Рис. 32.*** *Пример экземпляра метамодели (абстрактной программы) мини-языка множеств*

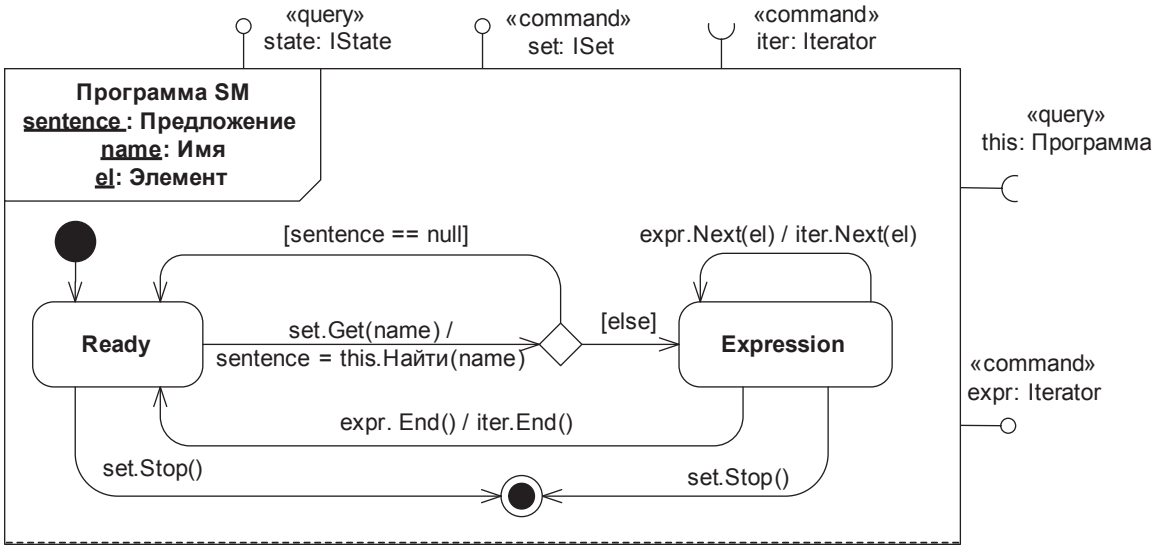


■ ***Рис. 33.*** *Экземпляры автоматов семантики и их взаимосвязь (пример 2)*

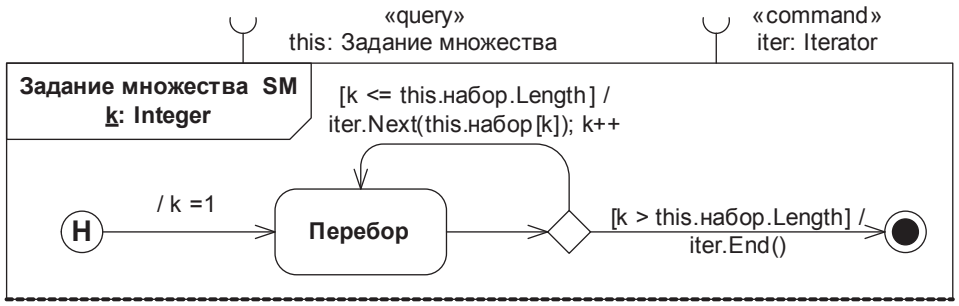
лировании на UML [4]. Эта диаграмма наглядно отображает систему объектов и связей между ними, находящуюся в памяти компьютера в процессе выполнения.

На рис. 33 объектами являются экземпляры автоматов, классы которых описаны на рис. 34–37. Заметим, что экземпляры автоматов появились только для «интерпретируемых» экземпляров классов Программа, Операция и Задание множества. Остальные классы, т. е. классы Предложение и Множество, используются только для установления связей между интерпретируемыми объектами. Эти связи учитываются в алгоритме CreateAuto и отражаются в схеме взаимосвязей автоматов неявным образом.

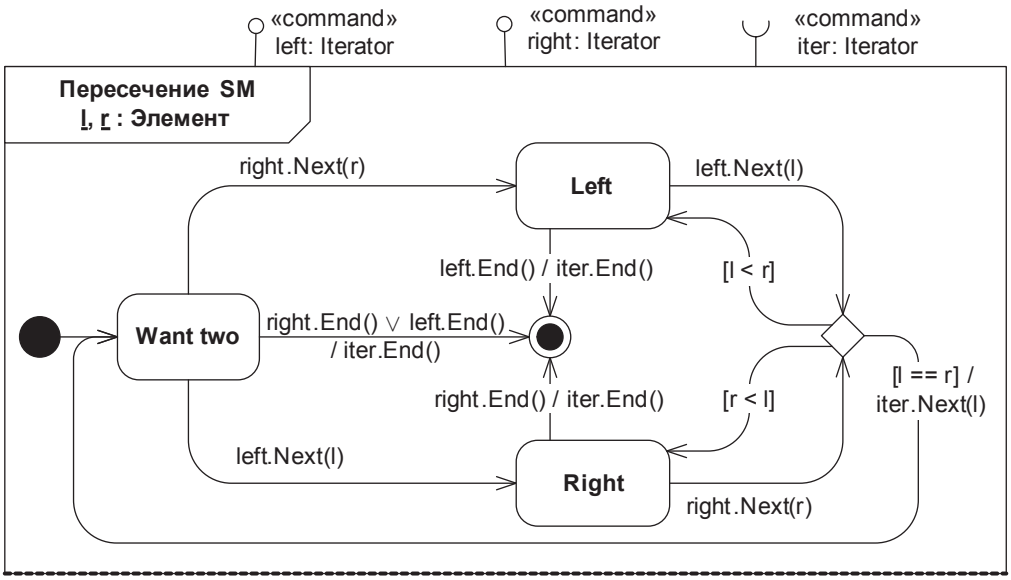
Головной автомат семантики мини-языка множеств (см. рис. 34) отличается от автомата на рис. 26 тем, что состояние Expression является простым, поэтому не используются точки выхода. Кроме того, у этого автомата есть интерфейс expr: Iterator, через который автомат принимает команды от итератора непосредственно интерпретируемого выражения.



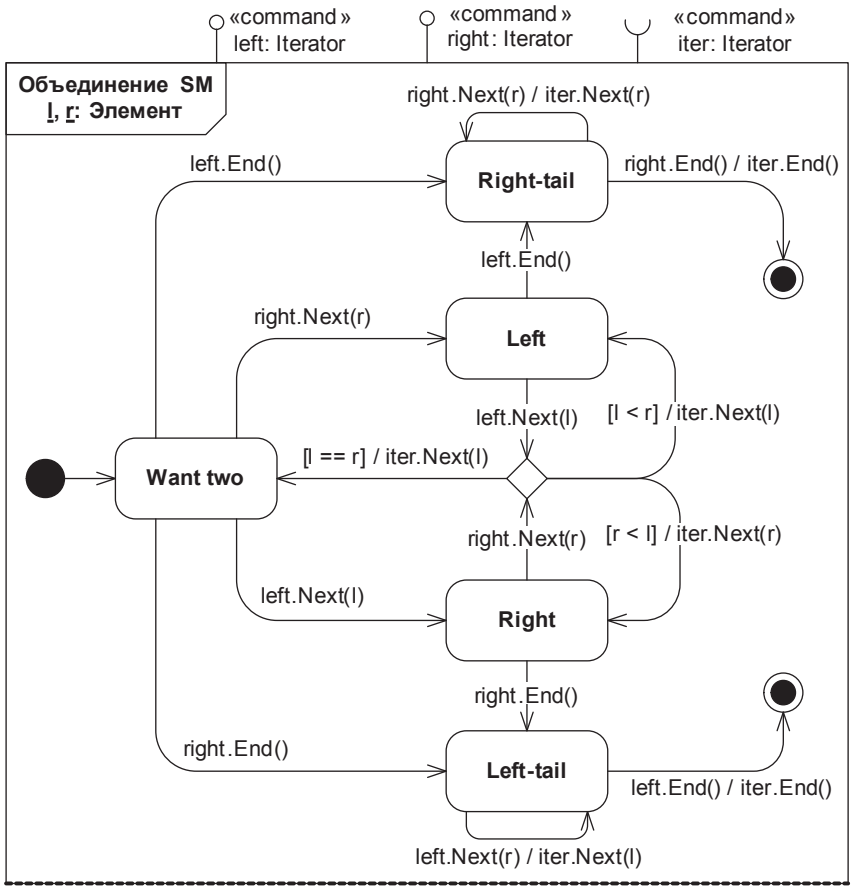
■ ***Рис. 34.*** *Головной автомат семантики мини-языка множеств (пример 2)*



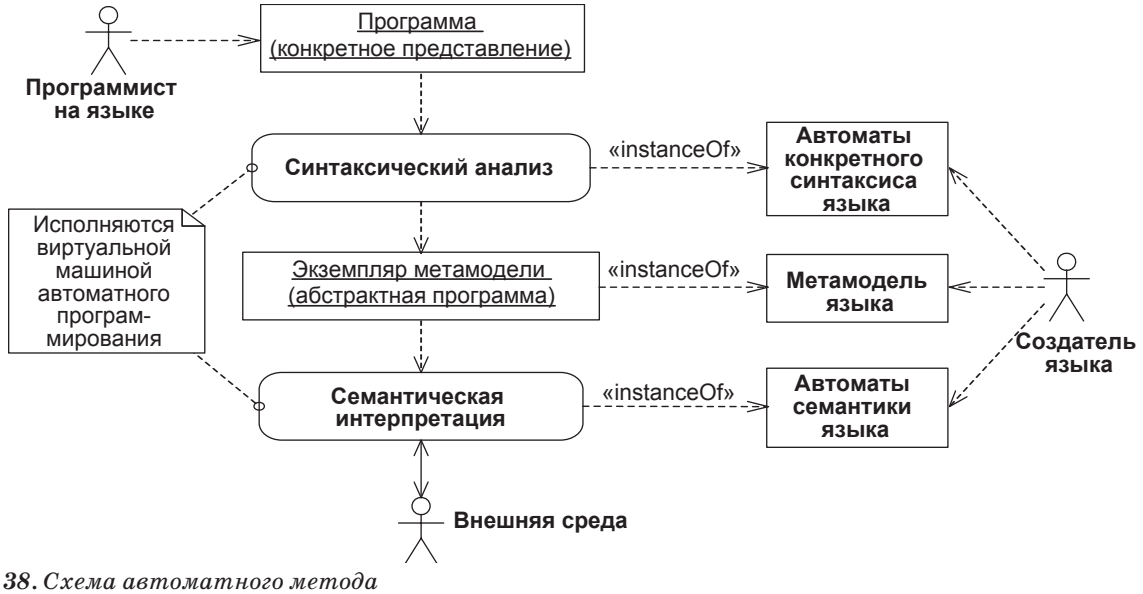
■ ***Рис. 35.*** *Автомат интерпретации задания множества в мини-языке множеств (пример 2)*



■ ***Рис. 36.*** *Автомат интерпретации операции пересечения в мини-языке множеств (пример 2)*



■ ***Рис. 37.*** *Автомат интерпретации операции объединения в мини-языке множеств (пример 2)*



■ ***Рис. 38.*** *Схема автоматного метода*

Работу автомата для задания множества перечислением элементов (см. рис. 35), в отличие от аналогичного автомата на рис. 28, необязательно прекращать каждый раз после получения очередного элемента. Кроме того, у него нет переменной el. Очередной элемент посылается в виде команды через интерфейс итератора, в эффекте на петле у состояния Перебор. По нашему мнению, автомат на рис. 35 намного естественнее и понятнее, чем на рис. 28.

Автомат, представленный на рис. 36, интерпретирует операцию пересечения множеств. Здесь не нужно заводить никаких флагов (сравн. рис. 29). Вся необходимая информация хранится в состояниях, как это и рекомендуется делать в автоматном программировании [5].

Наконец, на рис. 37 приведен автомат, интерпретирующий операцию объединения множеств. При сравнении этого автомата с автоматом на рис. 30 бросается в глаза, что хотя здесь состояний больше, но автомат намного понятнее. Нет сегментированных переходов, флагов и лишних локальных переменных.

Приведенный пример, на наш взгляд, убедительно показывает преимущества и гибкость выбранной автоматной модели.

Подводя итог обсуждению семантики, представим обобщенную модель автоматного метода в форме неканонической диаграммы, показывающей взаимосвязи основных составляющих метода (рис. 38).

Заключение

В статье предложен и подробно описан новый автоматный метод определения проблемно-ориентированных языков, базирующийся на описании структуры с помощью диаграмм классов и описании поведения с помощью расширенных диаграмм автомата UML. Отправной точкой создания и использования проблемно-ориентированного языка в автоматном методе является определение метамодели. На основе метамодели с помощью унифицированного метода описания поведения расширенными диаграммами автомата строятся множества синтаксических представлений и семантических интерпретаций. Это поведение реализуется виртуальной машиной автоматного программирования, что позволяет использовать описание языка как его реализацию. Основным преимуществом автоматного метода является гибкость: событийная модель позволяет использовать произвольные представления программ, а не только тексты; вариативная модель взаимодействия между автоматами позволяет выбрать адекватный стиль описания семантики языка.

Литература

1. **Грис Д.** Конструирование компиляторов для цифровых вычислительных машин. — М.: Мир, 1975. — 540 с.
2. **Новиков Ф. А., Тихонова У. Н.** Автоматный метод определения проблемно-ориентированных языков. Ч. 2 // Информационно-управляющие системы. 2010. № 2. С. 29–37.
3. **Новиков Ф. А.** Дискретная математика для программистов: учебник для вузов. 3-е изд. — СПб.: Питер, 2009. — 384 с.
4. **Новиков Ф. А., Иванов Д. Ю.** Моделирование на UML. Теория, практика, видеокурс. — СПб.: Наука и техника, 2010. — 640 с.
5. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. — СПб.: Питер, 2008. — 177 с.