МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

DOI 10.17223/20710410/15/7

УДК 004.432:004.435

ЯЗЫК ПРОГРАММИРОВАНИЯ AspectTalk

Д. А. Стефанцов

Национальный исследовательский Томский государственный университет, г. Томск, Россия

E-mail: d.a.stefantsov@isc.tsu.ru

Описывается язык объектно-ориентированного (ООП) и аспектно-ориентированного (АОП) программирования AspectTalk, состоящий из базового языка, метаязыка, библиотек ООП и АОП. Он реализуется с помощью метапрограммирования, протоколов метаобъектов и механизма примесей. Приводится сравнение AspectTalk с языками программирования, имеющими похожие возможности.

Ключевые слова: *язык программирования, метапрограммирование, Smalltalk, ООП, АОП, метаобъектный протокол.*

Введение

Язык AspectTalk разработан для проведения экспериментальных исследований в области защиты программных систем обработки информации путём их интеграции с политиками безопасности средствами аспектно-ориентированного программирования [1 – 3].

Будучи языком метапрограммирования, язык АОП может быть построен на базе любого языка программирования. В качестве базового языка в AspectTalk используется собственный диалект языка объектно-ориентированного программирования Smalltalk, поскольку, во-первых, в настоящий момент ООП является наиболее распространённой парадигмой программирования, а во-вторых, язык Smalltalk имеет самый маленький набор правил записи среди существующих языков ООП.

В ООП вычисления производятся посылкой сообщений от одних объектов другим с просьбой произвести некоторые вычисления. При этом каждый объект обладает набором сообщений, на которые он может ответить. Этот набор называется *протоколом* объекта. Посылка сообщений — это способ вызова подпрограммы в объектной системе. Подробнее о системах ООП можно прочитать, например, в [4].

Свойства ООП и АОП в языке AspectTalk реализуются с помощью метаязыка, модули на котором изменяют семантику конструкций базового языка. Библиотека АОП реализована в два этапа: на первом — с помощью метаязыка в объектную систему вводятся метаобъекты, на втором — с помощью протоколов метаобъектов описывается механизм примесей.

Любые вычисления можно построить только на посылке сообщений, по примеру лямбда-исчисления [5], но в реальной вычислительной системе есть потребности взаимодействия с внешними устройствами — дисками, клавиатурой, монитором, мышью и т. д., а также в эффективных вычислениях и в управлении памятью. Для того чтобы

удовлетворить эти потребности, в объектную систему вводится дополнительная сущность — *виртуальная машина* (ВМ), выполняющая требуемый набор операций, называемых, по традиции, *примитивами*.

Для упрощения процесса трансляции и выполнения программы ВМ представляют, как правило, не набором подпрограмм, а в виде самостоятельной программы, принимающей на вход последовательности команд. Допустимые для ВМ команды образуют *язык ВМ*.

В данной работе описывается язык AspectTalk: подробно — его базовый язык и метаязык и кратко — библиотеки ООП и АОП. Последняя реализована с помощью протоколов метаобъектов. Сообщается также об используемых ВМ и о трансляторе с AspectTalk в промежуточный язык. Показано, как с помощью основных конструкций языка AspectTalk можно реализовать свойства распространённых объектно-ориентированных и аспектно-ориентированных систем.

1. Базовый язык

Объекты представляют собой наборы данных и допустимых операций над этими данными. Данные и операции, как правило, подбираются способом, подходящим для той или иной предметной области. Например, есть объекты-строки, представляющие собой последовательности символов. Операции над объектами-строками — это конкатенация, взятие подстроки, поиск подстроки в строке и т. д. Есть также объекты-числа с допустимыми операциями — сложением, умножением, делением, вычитанием и т. д. Данные объекта и операции над объектом называются соответственно *член-функциями* и *член-данными* объекта.

Объектная система строится рекурсивно: данные объекта являются ссылками на другие объекты. Для упрощения вводятся *элементарные данные* — объекты, реализация которых скрыта от пользователя объектной системы (программиста) с помощью ВМ. Элементарные данные могут не содержать ссылок на другие объекты.

Существует несколько способов задания множеств операций над объектами. Алгоритм поиска операции по её имени называется *диспетчеризацией*.

Член-данные и член-функции объекта хранятся в *словарях* — объектах, являющихся важными компонентами любой объектной системы. Словарь представляет собой функцию, отображающую множество одних объектов в множество других. Например, словарь может отображать числа на строки, содержащие записи этих чисел в восьмеричной системе счисления, и быть определённым только для чётных чисел. Он может отображать строки на числа и другие словари и быть определённым только для строк «мой дядя», «самых», «честных правил». Аргументы функции, представляемой словарём, называются *ключами*. Пара (ключ, значение) в словаре называется *элементом* этого словаря. Операции над словарями позволяют получить значение, соответствующее некоторому ключу, изменить значение, соответствующее некоторому ключу (т. е. расширить функцию), добавить новый ключ в словарь (т. е. расширить функцию, представляемую словарём) и т. д.

По цели использования все словари объектной системы можно разделить на пользовательские и системные. Первые пользователь объектной системы (т. е. программист) явно задаёт в программе, вторые определяются в объектной системе неявно и используются для её работы. Например, множество переменных, используемых в подпрограмме, задаётся словарём, отображающим имена переменных в их значения.

Здесь описываются базовая часть языка AspectTalk, а также конструкции языка в порядке их усложнения. Для каждой конструкции приводится соответствующая

часть грамматики в бэкусовой форме. Лексемы языка вводятся по мере необходимости с помощью регулярных выражений.

1.1. Литералы

Базовый язык AspectTalk включает стандартные типы объектов, необходимые для реализации большинства алгоритмов, — числа, строки, массивы и т. п. Для описания объектов стандартных типов в языке AspecTalk существуют специальные конструкции, называемые *литералами* и обладающие следующим свойством: два объекта, описанные одной конструкцией, равны. Как правило, строковое представление объекта стандартного типа, используемое, например, при выводе этого объекта на экран, совпадает с литералом, описывающим данный объект (листинг 1).

```
1 integer   ::= [+\-]?([0-9]+r)?[0-9a-zA-Z]+
2 string    ::= '([^']|'')+'
3 char      ::= \$.
4 symbol    ::= #[^\t\n\r [\]{}()"'#;.]+
5 <literal> ::= integer | string | char | symbol | <array>
6 <array>   ::= "#(" <literal> ")"
```

Листинг 1. Литералы языка AspectTalk

Числа

На данный момент язык AspecTalk поддерживает только литералы целых чисел. Соответствующее регулярное выражение представлено в строке 1 листинга 1. Целое число начинается с необязательного знака, после чего следует необязательное указание на систему счисления (по умолчанию используется десятичная), а затем записывается модуль числа.

Основание системы счисления может принимать значения от 2 до 36 включительно. Для записи цифр больше 9 используются символы английского алфавита в верхнем или нижнем регистре.

Примеры литералов, задающих целые числа: `+0`, `2r0`, `16r0`, `-3r0`. Все перечисленные литералы задают число 0.

Строки

Строковые литералы представляют собой последовательности знаков, заключенные в одинарные кавычки (см. строку 2 листинга 1). Для того чтобы включить знак одинарной кавычки в строковый литерал, его необходимо продублировать.

Примеры строковых литералов: `'мой дядя'`, `'самых'`, `'честных правил'`, `'Dot your i''s and cross your t''s'`.

Знаки

Знаки являются составными частями строк и задаются литералами, удовлетворяющими регулярному выражению в строке 3 листинга 1: литерал, задающий знак, начинается с символа «$», после которого следует требуемый знак.

Например, строка `'мой дядя'` содержит знаки `$м`, `$о`, `$й`, `$ `, `$д`, `$я`, `$д`, `$я`.

Символы

Символы используются в качестве ключей в системных словарях, т. е. в словарях член-функций, член-данных, аргументов, локальных переменных. Ключи в словарях член-функций называются *селекторами*, в прочих системных словарях — *идентификаторами*. Регулярное выражение для литералов, описывающих символы, представ-

лено в строке 4 листинга 1: описание символа начинается со знака «#», после которого записывается последовательность знаков. Примеры символов: `#size`, `#at:put:`.

Массивы

В языке AspectTalk, как и в языке Smalltalk, существуют литералы для массивов, элементы которых могут быть заданы литералами. Описание массива начинается со знаков «`#(`», после чего следуют 0 или более литералов, разделённых пробельными знаками, затем следует знак «`)`». В строках 5 и 6 листинга 1 приведена бэкусова форма, определяющая возможные литералы для описания массивов; внутри массивов знаки «#» могут быть опущены при описании символов и вложенных массивов.

Примеры массивов: `#()`, `#(1 #two (three) 'four')`.

Во многих языках программирования литералы отождествляются с константами. В языке AspectTalk этого отождествления следует избегать, поскольку массив, заданный литералом, не является константой и может быть изменён.

1.2. Переменные

В объектной системе объекту может быть сопоставлено имя. Соответствие имени объекту хранится в системных словарях. Элементы системных словарей, используемые для хранения данных, называются *переменными*. Будем говорить, что в переменной x *хранится объект* y или *значение* y, если объект y имеет имя x в объектной системе.

Имя переменной является идентификатором и представляет собой последовательность букв, цифр и знаков подчёркивания, начинающуюся не с цифры. Регулярное выражение, описывающее возможные идентификаторы, есть `[a-zA-Z_][0-9a-zA-Z_]*`. Примеры идентификаторов: `x`, `y`, `index`.

Переменные в языке AspectTalk хранят только ссылки на объекты, как, например, переменные в языках Smalltalk, Python и Java. Объекты хранятся в динамической памяти ВМ и удаляются механизмом сборки мусора в случае, если объект становится недоступным через текущие переменные программы. В отличие от языка Java, переменной не сопоставлена метка типа хранящегося объекта, поэтому в одной и той же переменной можно хранить объекты разных типов, как в языках Smalltalk и Python.

Присвоение значения переменной производится с помощью специальной операции, имеющей, по историческим причинам, три обозначения: «`:=`», «`<-`» и «`_`». В листинге 2 приведены примеры присвоений: в строке 1 переменной `x` присваивается значение `'Hello, world!'`, в строке 2 переменной `y` присваивается значение переменной `x`, в строке 3 производится множественное присвоение — в переменных `x` и `y` сохраняется ссылка на целое число `31337`.

```
1 x := 'Hello, world!'
2 y := x
3 y := x := 31337
```

Листинг 2. Примеры присвоений в языке AspectTalk

В языке AspectTalk есть псевдопеременные — переменные, значение которых всегда определено тем или иным образом и не может быть изменено командой присвоения. Примерами псевдопеременных являются `true`, `false` и `nil`, указывающие всегда на булевы константы 0, 1 и «пустой объект» соответственно.

1.3. Посылка сообщений

Объекты взаимодействуют при помощи посылки сообщений друг другу. Сообщение включает в себя имя и набор параметров. После получения сообщения объект

выполняет процедуру диспетчеризации — находит необходимый алгоритм обработки принятого сообщения. В процессе обработки сообщения объект может посылать сообщения другим объектам или самому себе. Заканчивается процесс обработки возвратом результата объекту, пославшему сообщение.

Результат процедуры диспетчеризации зависит от параметров трёх типов — объекта, принимающего сообщение, имени сообщения и параметров сообщения.

Процесс посылки сообщения можно рассматривать как способ вызова подпрограммы. Главные отличия: 1) позднее связывание сообщения с алгоритмом, выполняющим его обработку; 2) алгоритму обработки сообщения всегда передаётся дополнительный параметр — ссылка на объект, принявший сообщение, которая хранится в псевдопеременной `self`.

Унарные сообщения

Унарные сообщения не имеют параметров. Для посылки унарного сообщения необходимо после объекта-получателя записать селектор сообщения. В листинге 3 приведены примеры посылки унарных сообщений: в строке 1 массиву посылается сообщение с селектором `#size`, после чего результат — размер массива, число 3 — записывается в переменную `n`; в строке 2 числу `n` посылается сообщение с селектором `#factorial`, после чего результат — число 6 — записывается в переменную `x`.

```
1  n := #(one two three) size
2  x := n factorial
```

Листинг 3. Примеры посылки унарных сообщений

Далее для краткости сообщение с селектором `#x` будем называть сообщением `x`.

Бинарные сообщения

Бинарные сообщения — это сообщения с одним параметром, селектор которых записывается с помощью специальных знаков: «`+`», «`-`», «`*`», «`/`», «`%`», «`<`», «`<=`», «`=`», «`>=`», «`>`», «`~=`», «`==`», «`~~`», «`@`», «`,`». Получатель сообщения указывается перед селектором, а параметр — после. Бинарные сообщения введены в язык для записи математических операций в естественном виде. Примеры посылки бинарных сообщений приведены в листинге 4: в строке 1 объекту `1` посылается сообщение `+` с параметром `2`, а результату — объекту `3` — посылается сообщение `*` с параметром `3`, после чего результат — объект `9` — записывается в переменную `x`; в строке 2 объекту `9`, хранящемуся в переменной `x`, посылается сообщение `=` с параметром `9`, после чего результат — объект `true` — записывается в переменную `y`.

```
1  x := (1 + 2) * 3
2  y := x = 9
```

Листинг 4. Примеры посылки бинарных сообщений

Сообщения с ключевыми словами

Для посылки сообщений с ненулевым количеством параметров в языке AspectTalk реализованы сообщения с ключевыми словами. Ключевое слово записывается в виде идентификатора со знаком двоеточия на конце: `at:`, `and:`, `or:`. Селектор сообщения с ключевыми словами содержит столько ключевых слов, сколько сообщение содержит параметров: `#at:put:`, `#to:do:`, `#ifTrue:ifFalse:`. В записи посылки сообщений с ключевыми словами сначала указывается объект-получатель, а затем записывается последовательность ключевых слов, после каждого из которых следует параметр.

В листинге 5 приведены примеры посылки сообщений с ключевыми словами: в строке 1 массиву посылается сообщение `at:` с параметром `2`, результатом является второй элемент массива — символ `#two`; в строке 3 массиву из пяти элементов, хранящемуся в переменной `y`, посылается сообщение `at:put:` с параметрами `5` и `0`, в результате чего в пятый элемент массива записывается число `0`.

```
1 x := #(one two three) at: 2
2 y := #(0 0 0 0 1)
3 y at: 5 put: 0
```

Листинг 5. Примеры посылки сообщений с ключевыми словами

Данный синтаксис записи сообщений с ключевыми словами отличается от наиболее распространённого варианта записи вызова метода с несколькими параметрами в таких языках, как C++, Java, Python, где имя сообщения записывается слитно, после чего следуют параметры в скобках через запятую. Главной целью введения такой формы записи в язык AspectTalk, как и в язык Smalltalk и, например, в Objective-C, является повышение читаемости программы.

Порядок вычисления выражений

В записях посылок сообщений вместо любого объекта может, в свою очередь, стоять выражение, результатом обработки которого является объект. Записи такого вида будем называть *выражениями*. Вложенные выражения анализируются с учётом приоритетов посылки сообщений. Порядок посылки сообщений в выражении в языке AspectTalk позаимствован из языка Smalltalk и определяется следующими правилами:

1) приоритеты типов сообщений в порядке убывания:
   а) унарные;
   б) бинарные;
   в) с ключевыми словами;
2) выражения для объектов получателя сообщения и параметров сообщения вычисляются в произвольном порядке;
3) сообщения с одинаковым приоритетом посылаются в порядке слева направо;
4) порядок вычисления выражения можно изменить с помощью скобок `(` и `)`.

В листинге 6 иллюстрируется порядок вычисления выражений:

1) в строке 2 приведено выражение, иллюстрирующее приоритет операций: наименее приоритетной операцией является посылка сообщения `at:put:` — она будет выполнена последней, при этом получатель (объект `x`) и параметры (`1` и результат выражения `x size + 1`) вычисляются в произвольном порядке, возможно, зависящем от транслятора; вычисление выражений `x` и `1` тривиально; в выражении `x size + 1` посылка бинарного сообщения `+` — наименее приоритетная операция, она будет выполняться последней, при этом получатель и параметр (`x size` и `1`) вычисляются в произвольном порядке;
2) в строке 3 приведено вычисление математического выражения с помощью бинарных селекторов; поскольку у бинарных селекторов одинаковый приоритет, посылка соответствующих сообщений начинается слева направо, т. е. сначала будет послано сообщение `+` объекту `1` с параметром `2`, затем результату (объекту `3`) будет послано сообщение `*` с параметром `3`; в данном случае приоритет операций языка расходится с приоритетом операций в математических записях; эта особенность взята из языка Smalltalk, у которого язык AspectTalk заимствует большинство свойств;

3) в строке 4 иллюстрируется использование скобок для изменения порядка вычисления выражения: второй элемент массива `x` увеличивается на единицу.

```
1 x := #(1 2 3 4 5)
2 x at: 1 put: x size + 1
3 y := 1 + 2 * 3
4 x at: 2 put: (x at: 2) + 1
```

Листинг 6. Примеры, иллюстрирующие порядок вычисления выражений

Каскады сообщений

В программах на AspectTalk, так же как и в программах на Smalltalk, часто возникает необходимость послать несколько сообщений одному и тому же объекту. Для этого вводится специальная синтаксическая конструкция — *каскады сообщений*. Для описания каскада необходимо после описания объекта (литерала, переменной или сообщения с посылками сообщений) перечислить посылаемые сообщения с параметрами, каждое из которых необходимо предварить знаком «`;`». Результатом вычисления выражения с каскадом всегда является объект, принимающий сообщения каскада.

В листинге 7 представлен пример использования каскадов сообщений. В этом примере словарь используется для хранения сведений о человеке: в строке 1 результатом выражения `Dictionary new` является новый словарь, которому в строках 2–4 посылаются три сообщения `at:put:`, сохраняющие в нём данные о человеке. Результат вычисления каскада — новый словарь, в который добавлено три элемента, — помещается в переменную `person`.

```
1 person := Dictionary new;
2     at: 'name' put: 'Иван Иванов';
3     at: 'age' put: 42;
4     at: 'height' put: 178
```

Листинг 7. Пример использования каскадов сообщений

Следует отметить, что каскадные выражения отличаются в диалектах языка Smalltalk — Smalltalk-80 и Little Smalltalk. В языке AspectTalk используются каскадные выражения из последнего.

1.4. Блоки

В языке AspectTalk выражения, состоящие из литералов, переменных, операций присвоения значения, посылок сообщений и каскадов, могут выполняться последовательно. При этом каждое из предложений выполняется ради побочного эффекта — изменения значения переменной, вывода строки на экран и т. п. Предложения отделяются друг от друга знаком точки «`.`». Точка после последнего предложения не обязательна. Пример последовательности предложений показан в листинге 8. Всякая программа на AspectTalk является последовательностью предложений.

```
1 s := 'Мой дядя самых честных правил'.
2 w := $я.
3 r := 0.
```

Листинг 8. Последовательность предложений на языке AspectTalk

В языке AspectTalk, как и в языке Smalltalk, можно описать объекты, представляющие собой последовательности предложений и называемые *блоками*. Как и всякие

объекты, блоки обладают состоянием и могут принимать и обрабатывать сообщения. Одним из таких сообщений является сообщение, запускающее выполнение предложений, содержащихся в блоке. Таким образом, блоки представляют собой разновидность подпрограмм. Параметры и результат сообщения, запускающего выполнение предложений блока, являются соответственно входными и выходным параметрами подпрограммы. С другой стороны, блоки являются объектами и могут быть сохранены в переменной, переданы в качестве параметров в другой блок, возвращены в качестве результата вычисления другого блока, а также могут принимать сообщения.

Описание блока начинается со знака «[», после которого следует необязательное описание входных параметров блока: имя каждого параметра предваряется знаком «:», параметры отделяются пробельными символами, описание параметров заканчивается знаком «|». После необязательного описания параметров записываются предложения блока. Описание блока заканчивается знаком «]». Блоки могут быть вложенными, т. е. содержать описания других блоков.

Если блок не имеет параметров, сообщение, запускающее выполнение предложений блока, имеет селектор `#value`. В противном случае селектор состоит из ключевых слов `value:`, количество которых равно числу параметров блока. Результат последнего предложения считается результатом выполнения всего блока.

В листинге 9, строка 1, показано описание блока с одним параметром и одним предложением, возвращающим значение `true`, если строка-параметр имеет ненулевую длину, и `false` в противном случае. В строке 2 запускается выполнение блока и его результат записывается в переменную `x`.

```
1 notEmpty := [ :str | str size > 0 ].
2 x := notEmpty value: s.
3 lcm := [ :n :m | d := n gcd: m. (n * m) / d ]
```

Листинг 9. Блок с одним параметром

Предложения блока могут содержать локальные переменные. В строке 3 листинга 9 определён блок, вычисляющий наименьшее общее кратное двух чисел. В процессе вычисления используется локальная переменная `d`, хранящая наибольший общий делитель чисел. Всякая запись значения в переменную, не содержащуюся в текущих системных словарях, добавляет эту переменную в словарь локальных переменных. Переменные в AspectTalk могут вводиться по необходимости во время своего первого использования. Словарь локальных переменных очищается при каждом запуске блока.

Поиск переменных всегда начинается со словаря локальных переменных текущего блока, продолжается в словаре параметров блока, после чего поиск рекурсивно продолжается среди переменных объемлющего блока. Таким образом, поиск переменных производится в цепочке словарей, начинающейся со словаря локальных переменных и заканчивающейся словарём глобальных переменных. В случае, когда переменная была найдена в одном из объемлющих блоков, указатель на эту переменную сохраняется в текущем блоке-объекте и хранится в нём до тех пор, пока он не будет удалён сборщиком мусора. Благодаря этой особенности, блоки в AspectTalk являются *замыканиями* в терминах таких языков, как Scheme и Python. Замыкания также реализованы в одной из популярных библиотек языка C++ [6].

В листинге 10 приведён пример программы, демонстрирующей возможности замыканий. В блоке `gen` объявляется локальная переменная `n` и инициализируется нулём, после чего блок возвращает вложенный блок, увеличивающий `n` на 1 и возвращающий значение `n`. Поскольку словарь локальных переменных блока очищается при каждом

запуске блока, указатели на переменные `n` в блоках `c1` и `c2` будут указывать на разные области памяти. Следовательно, `c1` и `c2` будут изменять разные области памяти и вести независимый счёт. Комментарии в строках 4–6 описывают значение соответствующих переменных.

```
1 gen := [ n := 0. [ n := n + 1. n ] ].
2 c1 := gen value.
3 c2 := gen value.
4 x := c1 value. " x = 1 "
5 y := c1 value. " y = 2 "
6 z := c2 value. " z = 1 "
```

Листинг 10. Замыкания в языке AspectTalk

Замыкания являются мощным механизмом, позволяющим кратко описывать сложные алгоритмы и структуры данных. В частности, библиотека ООП, описываемая далее, реализована с помощью замыканий.

Условные переходы

В языке AspectTalk, как и в языке Smalltalk, нет специальных синтаксических конструкций для определения условных переходов. Для реализации ветвлений в программе используются блоки и булевы значения `true` и `false`. Последним могут быть посланы сообщения с селекторами `#ifTrue:`, `#ifFalse:`, `#ifTrue:ifFalse:`, `#ifFalse:ifTrue:`, принимающие блоки в качестве параметров и запускающие их выполнение в зависимости от значения, при этом результат сообщения равен результату запущенного блока или `nil`, если не был запущен ни один блок (см. таблицу).

Ветвления в языке AspectTalk

Селектор сообщения	`true`	`false`
`#ifTrue:`	Запускается блок-параметр	`nil`
`#ifFalse:`	`nil`	Запускается блок-параметр
`#ifTrue:ifFalse:`	Запускается 1-й блок-параметр	Запускается 2-й блок-параметр
`#ifFalse:ifTrue:`	Запускается 2-й блок-параметр	Запускается 1-й блок-параметр

В листинге 11 приведены примеры условного перехода. В строке 1 первый знак строки `s` сравнивается со знаком, хранящимся в переменной `w`; в случае совпадения переменная `r` увеличивается на 1. В строках 2–4 устанавливается взаимная простота чисел `n` и `m`. Если значение выражения `(n gcd: m) = 1` равно `true`, то переменной `x` присваивается значение `1`, а если `false`, то `0`.

```
1 (s at: 1) = w ifTrue: [ r := r + 1].
2 (n gcd: m) = 1
3     ifTrue: [ x := 1 ]
4     ifFalse: [ x := 0 ].
```

Листинг 11. Пример условного перехода в языке AspectTalk

Циклы

Как и в случае условных переходов, в языке AspectTalk нет специальных синтаксических конструкций для организации циклов — они реализуются с помощью блоков.

Самый простой в реализации вид циклов — циклы с условием. Цикл с условием реализуется посылкой сообщения `#whileTrue` блоку. Для обработки этого сообщения блок вычисляет сам себя (посылая сообщение `#value` объекту в псевдопеременной `self`), и

если результат — это объект `true`, то рекурсивно посылает себе сообщение `#whileTrue`. Рекурсия продолжается до тех пор, пока значение, возвращаемое в ответ на сообщение `#value`, равно `true`. Следовательно, цикл с условием можно описать следующим образом: тело цикла записывается в виде предложений некоторого блока без параметров; последнее предложение представляет собой условие продолжения цикла; описанному блоку посылается сообщение `#whileTrue`. Очевидно, реализуемый подобным образом цикл является циклом с постусловием.

Цикл с предусловием реализуется с помощью сообщения `#whileTrue:` — блок-приёмник сообщения вычисляет себя (`self value`), и если результат равен `true`, вычисляет блок-параметр, после чего рекурсивно посылает себе сообщение `#whileTrue:` с тем же параметром, который был получен из предыдущего сообщения. Условие продолжения цикла записывается в блоке-приёмнике, а тело цикла — в блоке-параметре.

Аналогичным образом обрабатываются сообщения `#whileFalse` и `#whileFalse:`; единственное отличие состоит в том, что проверяемое условие — это условие выхода из цикла, а не условие продолжения цикла.

В листинге 12 приведён пример программы на AspectTalk, подсчитывающей количество вхождений знака `$я` в строку `'Мой дядя самых честных правил`. Результат накапливается в переменной `r`. В данной программе используется сообщение `#ifTrue:`.

```
1 s := 'Мой дядя самых честных правил'.
2 w := $я.
3 r := 0.
4
5 [ s size > 0 ] whileTrue:
6     [ (s at: 1) = w
7           ifTrue: [ r := r + 1 ].
8       s := s allButFirst. ].
```

Листинг 12. Пример использования цикла с условием

Несмотря на то, что циклы в AspectTalk реализуются с помощью рекурсии, их выполнение не приводит к росту стека вызовов подпрограмм, так как в их реализации используется механизм хвостовой рекурсии, описанный в [7].

Очевидно, цикл со счётчиком можно реализовать с помощью цикла с условием, заведя переменную цикла, увеличивая её на каждой итерации и сравнивая с конечной границей. Данная дисциплина организации цикла для удобства реализована в виде сообщений `#to:do:` и `#to:by:do:`, посылаемых целым числам: получатель является начальной границей счётчика, параметр ключевого слова `to:` — конечной границей счётчика, параметр ключевого слова `do:` — телом цикла. В сообщении `#to:by:do:` есть также возможность указать шаг счётчика с помощью параметра `by:`. Необходимо отметить, что счётчик передаётся в блок — тело цикла при вычислении последнего.

В листинге 13 приведён тот же алгоритм, что и на листинге 12, но с использованием цикла со счётчиком вместо цикла с условием.

```
1 s := 'Мой дядя самых честных правил'.
2 w := $я.
3 r := 0.
4 1 to: s size do:
5     [ :i | (s at: i) = w ifTrue: [ r := r + 1 ] ].
```

Листинг 13. Пример использования цикла со счётчиком

В языке AspectTalk есть возможность организации циклов по элементам объектов-агрегатов, содержащих в себе множество равнозначных ссылок на другие объекты. Примерами объектов-агрегатов являются массивы, строки и словари. Для описания подобного цикла объекту-агрегату посылается сообщение `#do:`, параметром которого является блок — тело цикла. В процессе обработки сообщения объект-агрегат перебирает содержащиеся в нём ссылки на объекты и для каждого из них посылает сообщение `#value:` телу цикла, передавая в качестве параметра текущий элемент.

В листинге 14 представлен тот же алгоритм, что и в листингах 12 и 13, но реализованный с помощью цикла по строке, являющейся агрегатом объектов-знаков.

```
1 s := 'Мой дядя самых честных правил'.
2 w := $я.
3 r := 0.
4
5 s do: [ :c | c = w ifTrue: [ r := r + 1 ] ].
```

Листинг 14. Пример использования цикла по объектам-агрегатам

1.5. Примитивы

Примитивом называется операция обращения к виртуальной машине. Примитивы используются для выполнения простых действий, таких, как арифметические операции над целыми числами, вывод строки на экран и т. п. Некоторые операции, которые могут быть выполнены средствами языка AspectTalk, реализуются виртуальной машиной в целях оптимизации. К таким операциям относится, например, преобразование числа в строку.

В отличие от Smalltalk, примитивы в AspectTalk имеют явные параметры и возвращаемое значение. В Smalltalk параметры и возвращаемое значение примитивов берутся из текущего стека, что затрудняет понимание текста программы, а также делает язык зависимым от реализации ВМ.

Примитивы можно разбить на три группы:

1) примитивы общего назначения;
2) примитивы, используемые для ввода и вывода;
3) примитивы, относящиеся к тому или иному типу данных.

В листинге 15 проиллюстрированы примитивы этих трёх групп: в строке 1 указан примитив `newObject`, результатом выполнения которого является новый объект; примитив `write` принимает два параметра — имя потока вывода (`stdout`) и строку, выводит переданную строку на поток вывода и возвращает `nil` (см. строку 2); в строке 3 используется примитив `integerAddition`, принимающий на вход два числа и возвращающий их сумму.

```
1 <newObject>
2 <write stdout 'Hello, world!'>
3 <integerAddition x 1>
```

Листинг 15. Примеры примитивов в языке AspectTalk

1.6. Грамматика языка AspectTalk

В листинге 16 описаны все возможные лексемы языка в виде регулярных выражений (строки 1–11) и правила синтаксиса в бэкусовой форме (строки 13–29). Лексемы записаны в виде идентификаторов, нетерминалы грамматики — в виде идентификато-

ров, окружённых угловыми скобками «<» и «>». Простые (односимвольные и двухсимвольные) лексемы представлены в синтаксических правилах буквально и окружены знаками «"».

Выделение группы терминалов и нетерминалов знаками

1) «(» и «)?» означает, что эта группа может присутствовать и отсутствовать в выражении, задаваемом правилом;
2) «(» и «)*» означает повторение группы ноль или более раз;
3) «(» и «)+» означает повторение группы один или более раз.

Выше описана семантика всех приведённых в листинге 16 конструкций, за исключением команды возврата результата в строке 14, которая будет описана в п. 3.

```
 1 identifier   ::= [a-zA-Z_][0-9a-zA-Z_]*
 2 keyword      ::= [a-zA-Z_][0-9a-zA-Z_]*:
 3 binarysel    ::= +|-|*|/|%|<|<=|=|>=|>|~=|==|~~|@|,
 4 unarysel     ::= [a-zA-Z_][0-9a-zA-Z_]*
 5 integer      ::= [+\-]?([0-9]+r)?[0-9a-zA-Z]+
 6 symbol       ::= #[^\t\n\r [\]{}()"'#;.]+
 7 string       ::= '([^']|'')+'
 8 comment      ::= "([^"]|"")+"
 9 char         ::= \$.
10 delimiter    ::= [\t\r\n ]+
11 assign       ::= _|<-|:=
12
13 <statements> ::= <statement> ( "." <statement> )*
14 <statement>  ::= ( "^" )? <expression>
15 <expression> ::= identifier assign <expression>
16 <expression> ::= <cascaded>
17 <cascade>    ::= <simple> ( ";" <selectors> )*
18 <simple>     ::= <binary> ( keyword <binary> )*
19 <binary>     ::= <unary> ( binarysel <unary> )*
20 <unary>      ::= <primary> ( unarysel )*
21 <primary>    ::= <variable> | <literal> |
22                  <block> | <primitive> | "(" <cascade> ")"
23 <selectors>  ::= ( unarysel | binarysel <unary> |
24                    ( keyword <binary> )+ )+
25 <block> ::= "[" ( (":" identifier)+ "|")?  <statements> "]"
26 <literal>    ::= integer | symbol | string | char | <array>
27 <array>      ::= "#(" ( <literal> )* ")"
28 <primitive>  ::= "<" identifier ( <primary> )* ">"
29 <variable>   ::= identifier
```

Листинг 16. Грамматика языка AspectTalk

2. Метаязык

Метаязык используется для изменения семантики синтаксических конструкций языка. Инструкции метаязыка записываются в отдельных модулях и имеют собственные грамматические правила.

Содержание метаязыкового модуля в AspectTalk описывает синтаксические правила, с помощью которых транслятор определяет конструкции, семантику которых необходимо изменить. Правила определяются в виде, схожем с бэкусовой формой (ли-

стинг 17). В качестве элементов описания допускается использование нетерминалов и лексем, определённых в грамматике базового языка AspectTalk. Последним правилом в метаязыковом модуле должно быть правило вида «`<handler> ::= identifier`», в котором задаётся имя блока, вызываемого вместо всех конструкций, описанных с помощью правил модуля. Таким образом семантика этих конструкций заменяется на определённую в указанном блоке. Такой блок будем называть *обработчиком*.

```
 1 <rules> ::= <rule> <newline> <rules>
 2 <rule> ::= <item> "::=" <items> |
 3            "<handler>" "::=" identifier
 4 <items> ::= <ritem> <items> |
 5             <ritem> "|" <items> |
 6             "(" <items> ")?" |
 7             "(" <items> ")*" |
 8             "(" <items> ")+" |
 9             "(" <items> ")"
10 <ritem> ::= <item> | <item> "->" identifier
11 <item> ::= identifier | "<" identifier ">"
```

Листинг 17. Грамматика метаязыковых модулей на AspectTalk

В правилах метаязыковых модулей можно использовать конструкцию «`->`», позволяющую получить объект из контекста выполнения синтаксической конструкции (см. строку 10). После служебного слова «`->`» указывается идентификатор, который будет использован в словаре всех объектов из контекста выполнения синтаксической конструкции. Этот словарь будет передан в блок-обработчик при его вызове.

Если конструкции, описанные правилами метаязыкового модуля, имеют возвращаемое значение (т. е. если описывается выражение), то значение, возвращаемое блоком-обработчиком, возвращается вместо значения описанных конструкций.

Заметим, что метаязыковые конструкции языка Groovy [8] позволяют решать аналогичные задачи, однако вариант AspectTalk удобнее в использовании, так как позволяет декларативно объявить интересующие конструкции, не заставляя программиста работать с конкретными деревьями разбора, которые могут различаться в разных трансляторах.

2.1. Порядок выполнения метаязыковых модулей

Порядок выполнения языковых модулей задаётся либо в отдельном файле, либо в командной строке. Сначала загружаются метаязыковые модули в указанном порядке, потом — модули на базовом языке в указанном порядке.

3. Реализация ООП в AspectTalk

С помощью метаязыка в языке AspectTalk реализованы свойства, традиционные для объектно-ориентированных языков: инкапсуляция, наследование и полиморфизм. В отличие от своего предшественника, AspectTalk лишён декларативных конструкций, описывающих как член-данные и член-функции объектов, так и традиционную для объектно-ориентированных языков иерархию наследования. Все необходимые действия задаются конструкциями языка, описанными выше.

3.1. Инкапсуляция

С каждым объектом в языке AspectTalk связаны два словаря — член-данных и член-функций объекта. Для каждого объекта эти словари могут формироваться индивидуально. При этом говорят, что объект инкапсулирует член-данные.

В листинге 18 приведён пример объявления объекта добавлением к его словарю член-данных переменной с именем `word` и к словарю его член-функций элемента с именем `say`, который выводит на экран строку `'Hello, world!'` при выполнении.

```
1 a := Object new;
2     addVariable: #word;
3     addMethod: #say usingBlock:
4         [ word := 'Hello, world!'.
5           word writeln ].
```

Листинг 18. Объявление объекта

Теперь если послать объекту `a` сообщение `say`, то на экран выведется строка `'Hello, world!'`. В листинге 18 модификация словарей объекта производится с помощью заранее определённых для всех объектов сообщений `addVariable:` и `addMethod:usingBlock:`. Эти методы обращаются к соответствующим примитивам виртуальной машины, которая и производит модификацию словарей.

В описании грамматики языка в листинге 16 указана конструкция, не описанная до сих пор (см. строку 14) — команда возврата результата, начинающаяся со знака «`^`», за которым следует выражение. Эта команда возвращает значение указанного выражения в качестве результата обработки текущего сообщения. Если команда встретилась в блоке, не являющемся член-функцией некоторого объекта, то работа блока прерывается, управление передаётся блоку, запустившему данный, и производится проверка, является ли он член-функцией некоторого объекта, и т. д. до тех пор, пока очередной блок из стека вызова блоков не окажется член-функцией некоторого объекта. После этого значение указанного в команде выражения возвращается в качестве результата работы найденной член-функции.

Следует упомянуть об изменении порядка поиска переменных в случае ООП: словарь член-данных текущего объекта встраивается в цепочку словарей переменных сразу после словарей локальных переменных и параметров текущего блока.

3.2. Наследование

Для того чтобы избежать необходимости задания словарей для каждого объекта, используются специальные объекты, называемые *классами*. Класс — это объект, способный конструировать другие объекты и задавать для них часть словарей член-данных и член-функций. В листинге 18 использован класс `Object`, посылкой сообщения `new` которому конструируется новый объект. Именно в классе `Object` определены член-функции `addVariable:` и `addMethod:usingBlock:`, позволяющие модифицировать словари объекта.

В листинге 19 приведено определение класса на языке AspectTalk. Как и в случае с определением объекта, удобным средством описания оказывается механизм каскадов сообщений.

Определяется класс `Talkative`; все объекты, сконструированные с его помощью (далее такие объекты называются *экземплярами* соответствующего класса), будут понимать сообщение `say` и выводить строку `'Hello, world!'` в процессе их обработки. В строке 8 представлено объявление экземпляра класса `Talkative` — объекта `a`. Это делается посылкой сообщения `new` объекту-классу `Talkative` (так же, как в листинге 18 сообщение `new` посылается объекту-классу `Object`). При получении данного сообщения класс не только конструирует свой экземпляр, но и посылает ему сообщение

`init` перед возвратом его в вызвавший алгоритм. Поэтому метод с именем `init` может считаться аналогом конструкторов в языках C++ и Java.

```
1 Talkative := Class new;
2     addInstanceVariable: #word;
3     addInstanceMethod: #init usingBlock:
4         [ word := 'Hello, world!' ];
5     addInstanceMethod: #say usingBlock:
6         [ word writeln ].
7
8 a := Talkative new
```

Листинг 19. Определение класса на языке AspectTalk

Модификация словарей экземпляров класса производится с помощью сообщений `addInstanceVariable:` и `addInstanceMethod:usingBlock:`, которые создают соответственно член-данные и член-функции класса как объекта. В языках C++ и Java такие член-данные и член-функции называются статическими.

Для сокращения избыточности программ в языке AspectTalk используется механизм наследования, привычный для всех распространённых объектно-ориентированных языков программирования: член-данные и член-функции, объявленные в классе-родителе, наследуются дочерним классом и не требуют повторного объявления. В листинге 20 приведён пример описания унаследованного класса. Наследование происходит с помощью передачи параметра сообщению `new:`.

```
1 Memorizer := Class new: Talkative;
2     addInstanceMethod: #remember: usingBlock:
3         [ :what | word := what ].
4
5 a := Memorizer new
```

Листинг 20. Объявление класса-потомка

Экземпляры класса `Memorizer`, подобно экземплярам класса `Talkative`, выводят на экран содержимое член-данного `word`, однако позволяют модификацию этого член-данного с помощью посылки сообщения `remember:`.

Как и в языках Java и Smalltalk, все классы объединены в единую иерархию наследования с корнем — классом `Object`. Следует отметить, что, как и в языках-предшественниках Smalltalk-80 и Little Smalltalk, в языке AspectTalk нет множественного наследования, характерного для языков C++ и Python. Ещё одной особенностью AspectTalk по сравнению с языком C++ является тот факт, что все член-функции на этом языке являются «виртуальными» в терминологии языка C++. Это происходит потому, что все переменные в языке AspectTalk являются указателями, как и в языках Smalltalk-80, Little Smalltalk, Python, Java и многих других, в которых также все член-функции являются «виртуальными» в терминологии языка C++.

3.3. Полиморфизм

Тот факт, что все член-данные в языке AspectTalk являются «виртуальными», обусловливает свойство полиморфизма языка. Следует отметить также, что два объекта

совершенно разных классов могут быть переданы в один блок при условии, что оба они обрабатывают сообщения, посылаемые параметрам блока в его предложениях. Это свойство обусловливается отсутствием типизации переменных языка AspectTalk.

При необходимости установления принадлежности объекта классу используются сообщения `isKindOf:` и `isMemberOf:`, определённые для всякого объекта и принимающие класс в качестве параметра. В первом случае результат будет истинен, если переданный в качестве параметра класс встречается на пути от класса объекта-получателя до корня иерархии наследования. Во втором случае результат будет истинен только тогда, когда получатель является экземпляром класса, переданного в качестве параметра.

3.4. Стандартная библиотека классов

Стандартная библиотека классов языка AspectTalk содержит описание объекта `Metaclass`, метакласса `Class` и следующих стандартных классов: `Object`, `Block`, `Execution`, `Comparable`, `Integer`, `Symbol`, `Char`, `String`, `Array`, `UndefinedObject`, `Boolean`, `True`, `False`.

На рис. 1 изображена иерархия наследования классов стандартной библиотеки языка AspectTalk. Все прямые подклассы класса `Object` сгруппированы для краткости в один узел.

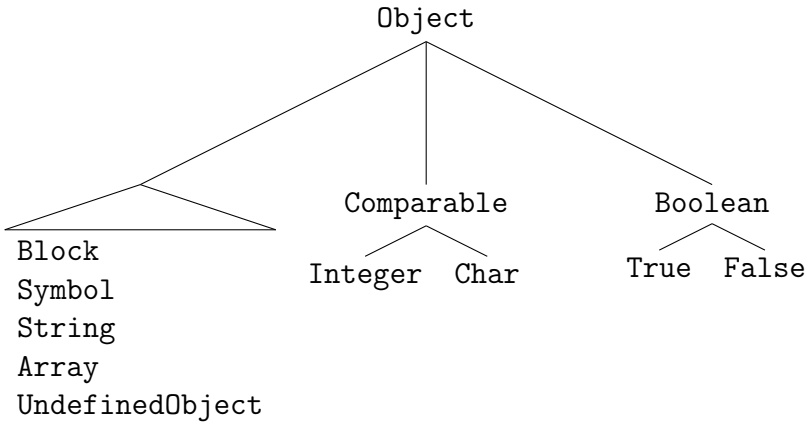

Рис. 1. Иерархия наследования классов стандартной библиотеки языка AspectTalk

4. Реализация АОП в AspectTalk

Аспектно-ориентированные свойства языка AspectTalk, как и его объектно-ориентированные свойства, описаны в библиотеке, реализованной с помощью метаязыка. Технология реализации АОП — протоколы метаобъектов [9]. Важным метаобъектом для реализации АОП является *метакласс*, который, во-первых, является классом класса, а во-вторых, модифицирует механизм передачи сообщений между объектами.

В листингах 19 и 20 уже использовался метакласс `Class`, посылкой сообщения `new` которому создаются объекты-классы. Это отражает первую особенность метаклассов: всякий класс в языке AspectTalk является экземпляром некоторого метакласса и принимает сообщения `addInstanceVariable:` и `addInstanceMethod:usingBlock:`, с помощью которых модифицируются словари экземпляров классов.

В листинге 21 отражена вторая особенность метаклассов — изменение механизма диспетчеризации сообщений. Если экземпляру класса, который, в свою очередь, является экземпляром метакласса `DialogueClass` либо подклассом экземпляра метакласса

`DialogueClass`, будет послано сообщение «`say`», то сначала будет исполнен блок, объявленный с помощью сообщения «`addEnvelope:usingBLock`», посланного метаклассу `DialogueClass`.

```
1 DialogueClass := Metaclass new;
2   addEnvelope: 'say' usingBlock: [ :sender :message :args |
3     '-- ' write.
4     ^ receiver call: message with: args.
5   ].
6 Talkative class: DialogueClass.
```

Листинг 21. Описание метакласса на языке AspectTalk

Этот блок имеет доступ ко всем член-данным объекта-получателя сообщения `say`; в качестве аргументов ему передаются объект-отправитель, селектор сообщения и массив, содержащий параметры сообщения. Псевдопеременная `receiver` ссылается на объект-получатель. Имея эти данные, можно повторно переслать сообщение (выполнить оригинальную член-функцию). В данном случае блок-конверт выводит на экран два знака «-» и пробел, после чего запускает выполнение оригинальной член-функции. Последняя строка листинга 21 меняет метакласс класса `Talkative` с `Class` на `DialogueClass`. В результате все сообщения, выводимые на экран экземплярами классов `Talkative` и `Memorizer`, будут предварены префиксом «`-- `».

Метаклассы, так же как и классы, организованы в иерархию наследования. При этом наследуются не только словари член-функций и член-данных, определённых для классов, но и словари конвертов, определённые для экземпляров этих классов.

Для защиты программных систем путём их интеграции с политикой безопасности (ПБ) посредством АОП предлагается использовать механизм примесей как альтернативу механизму множественного доступа. Этот метод защиты подробно изложен в [3]. Его суть следующая:

1) незащищённая программная система описывается в терминах классов;
2) реализация ПБ описывается в терминах примесей;
3) декларативным способом устанавливается соответствие классов и примесей, а также шаблоны для вызовов алгоритмов-конвертов.

Примеси могут быть описаны в библиотеке АОП с помощью метаклассов. На этапе трансляции происходит объединение классов и примесей в один общий, составной класс с изменённой диспетчеризацией сообщений с помощью алгоритмов-конвертов.

Похожий подход предложен в [10], но аспектно-ориентированная система у нас проще системы, описанной в [10], поскольку она не разделяет статические и динамические свойства системы и описывает оба типа соответствующих примесей единообразно.

5. Транслятор с AspectTalk в промежуточный язык

Технология ВМ широко используется в современном программировании. Существующие ВМ в достаточной мере покрывают потребности языка AspectTalk: 1) обладают возможностями управления памятью, в том числе имеют функцию «сборки мусора»; 2) обладают необходимым набором примитивных операций, в том числе над требуемыми элементарными данными; 3) выполнены в виде отдельной программы. В большинстве случаев языком ВМ является польская инверсная запись, задаваемая последовательностью команд ВМ.

Изначально для реализации AspectTalk была использована стековая ВМ. В настоящее время AspectTalk транслируется в язык Scheme [13], а в качестве ВМ используется интерпретатор Scheme. Использование Scheme в качестве промежуточного языка упрощает трансляцию модулей на метаязыке.

6. Сравнение с другими языками

По ходу изложения уже отмечались отдельные сходства и различия между языком AspectTalk и существующими языками программирования, имеющими метаязыковые конструкции. Перечислим основные из них.

По сравнению с AspectJ [11]:

1) упрощённый синтаксис;
2) явный порядок применения нескольких аспектов;
3) невозможность описания логических парадоксов [12];
4) аспекты, независимые от программы (в т. ч. в аспектной библиотеке).

По сравнению со Scheme [13]:

1) больше возможностей для императивного программирования, привычного для большинства программистов;
2) макросы не имеют системного имени; возможность переопределить базовые конструкции языка с помощью макросов.

В настоящее время AspectTalk транслируется в Scheme, поэтому программы, эквивалентные AspectTalk, можно писать сразу на Scheme. (Эта ситуация аналогична ситуации с первыми трансляторами с C++, которые переводили программы на C++ в программы на C.) При этом использование нового языка для обозначения часто используемых конструкций на другом языке имеет следующие преимущества:

1) AspectTalk поощряет АОП и метапрограммирование в целом;
2) он запрещает небезопасные использования конструкций.

По сравнению со Smalltalk [14]:

1) объектная модель не является частью языка (наследование, словари времени трансляции);
2) объектная библиотека AspectTalk не ограничивает использование метаклассов: есть возможность описывать метаклассы явно, порождать несколько классов от одного метакласса;
3) метапрограммирование не ограничивается работой с метаклассами, разрешены описания точек выполнения с помощью метаязыка;
4) некоторые конструкции базового языка AspectTalk имеют отличную от своих аналогов в Smalltalk семантику.

Стоит отметить также, что язык Python [15], как и AspectTalk, не ограничивает использование метаклассов, но, как и Smalltalk, имеет жёсткую объектную модель и ограниченные возможности метапрограммирования.

По сравнению с MetaclassTalk [10]:

1) аспектная библиотека AspectTalk похожа на АОП в MetaclassTalk;
2) в MetaclassTalk метаобъектный протокол — единственное средство метапрограммирования; метаязык AspectTalk предоставляет возможность реализации более мощной аспектной библиотеки.

По сравнению с Groovy [8]:

1) язык Groovy, как и язык AspectTalk, состоит из двух уровней: языка и метаязыка;
2) метаязык Groovy так же, как и метаязык AspectTalk, позволяет изменять семантику некоторых синтаксических конструкций, однако для этого необходима манипуляция деревом синтаксического разбора как структурой в языке Groovy; это осложняет реализацию метаязыковых модулей.

Заключение

Язык AspectTalk разработан для проведения экспериментальных исследований в области АОП. В основе языка лежат свойства языков Smalltalk, Scheme, MetaclassTalk, Python, AspectJ и Groovy, которые комбинировались и модифицировались для достижения простоты синтаксиса и семантики, отсутствия противоречий (невозможности описания логических парадоксов), возможности реализации АОП, удобного для описания политик безопасности и интеграции их с программами.

Свойства АОП и метапрограммирования, реализованные в AspectTalk, могут использоваться для реализаций аспектно-ориентированных версий других языков программирования. В частности, перспективным направлением представляется реализация JIT-транслятора, переводящего последовательность инструкций для архитектуры x86 в другую последовательность инструкций для этой же архитектуры, но содержащую в себе, помимо оригинальных команд, команды политики безопасности. Возможными средствами для реализации подобного транслятора являются библиотеки [16, 17].

За рамками данной работы остались формальное описание семантики языка AspectTalk и детальное описание его примитивов, стандартной библиотеки и реализации библиотек ООП и АОП в нём и транслятора с него в язык Scheme. Предполагается, что это будет сделано в последующей публикации автора.

ЛИТЕРАТУРА

1. *Стефанцов Д. А.* Реализация политик безопасности в компьютерных системах с помощью аспектно-ориентированного программирования // Прикладная дискретная математика. 2008. № 1(1). С. 94–100.
2. *Стефанцов Д. А.* Технология и инструментальная среда создания защищённых систем обработки информации // Прикладная дискретная математика. Приложение. 2009. № 1. С. 55–56.
3. *Стефанцов Д. А.* Внедрение политик безопасности в программные системы обработки информации // Прикладная дискретная математика. 2011. № 3(13). С. 55–64.
4. *Budd T. A.* An Introduction to Object-Oriented Programming. 3rd edition. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. 611 p.
5. *Revesz G. E.* Lambda-calculus, Combinators and Functional Programming. New York, NY, USA: Cambridge University Press, 2009. 192 p.
6. `www.boost.org/libs/lambda` — Boost C++ Libraries. Chapter 14. Boost. Lambda. 2009.
7. *Абельсон Х., Сассман Дж. Дж.* Структура и интерпретация компьютерных программ. М.: Добросвет, 2010. 608 с.
8. `http://groovy.codehaus.org/` — Groovy. A dynamic language for the Java platform. 2012.
9. *Kiczales G., des Rivières J., and Bobrow D. G.* The art of metaobject protocol. Cambridge, MA, USA: MIT Press, 1991. 345 p.

10. *Bouraqadi N., Seriai A., and Leblanc G.* Towards unified aspect-oriented programming // ESUG 2005 Research Conf. Brussels, Belgium, 2005. 22 p.
11. `http://www.eclipse.org/aspectj/` — AspectJ. Crosscutting objects for better modularity. 2012.
12. *Forster F. and Steimann F.* AOP and the antinomy of the liar // Workshop on the Foundations of Aspect-Oriented Languages. 2006. P. 47–56.
13. *Sperber M., Dybvig R. K., Flatt M., et al.* Revised[6] Report on the Algorithmic Language Scheme. New York, NY, USA: Cambridge University Press, 2010. 302 p.
14. *Goldberg A., Robson D., and Harrison M. A.* Smalltalk-80: The Language and its Implementation. Boston, MA, USA: Addison-Wesley, 1983. 714 p.
15. `http://python.org/` — Python Programming Language. 2012.
16. `http://www.pintool.org/` — PIN Toool. 2011.
17. `http://dynamorio.org/` — DynamicRIO. Dynamic Instrumentation Tool Platform. 2012.