МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

СОГЛАСОВАНО
Генеральный директор
ЗАО «АйТи»

_____ Бакиев О.Р.
"19" декабря 2011 г.

УТВЕРЖДАЮ
Ректор НИУ ИТМО

_____ Васильев В.Н.
"28" декабря 2011 г.

МНОГОПРОФИЛЬНАЯ ИНСТРУМЕНТАЛЬНО-ТЕХНОЛОГИЧЕСКАЯ ПЛАТФОРМА СОЗДАНИЯ И УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ СРЕДОЙ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ CLAVIRE

ПРОГРАММНЫЙ КОМПОНЕНТ РАЗБОРА СКРИПТА EASYFLOW CLAVIRE/EASYFLOW

ОПИСАНИЕ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.80066-06 13 19-ЛУ

Представители
Организации-разработчика

Руководитель разработки,
профессор НИУ ИТМО

_____ Бухановский А.В.
"28" декабря 2011 г.

Ответственный исполнитель,
с.н.с. НИУ ИТМО

_____ Луценко А.Е.
"28" декабря 2011 г.

Нормоконтролер
ведущий инженер НИУ ИТМО

_____ Позднякова Л.Г.
"28" декабря 2011 г.

2011

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДЕН
RU.СНАБ.80066-06 13 19-ЛУ

МНОГОПРОФИЛЬНАЯ ИНСТРУМЕНТАЛЬНО-ТЕХНОЛОГИЧЕСКАЯ ПЛАТФОРМА СОЗДАНИЯ И УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ СРЕДОЙ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ CLAVIRE

ПРОГРАММНЫЙ КОМПОНЕНТ РАЗБОРА СКРИПТА EASYFLOW CLAVIRE/EASYFLOW

ОПИСАНИЕ ПРОГРАММЫ

RU.СНАБ.80066-06 13 19

ЛИСТОВ 36

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

2011

АННОТАЦИЯ

Документ содержит описание программного компонента разбора скрипта EasyFlow CLAVIRE/EasyFlow RU.СНАБ.80066-06 01 19 многопрофильной инструментально-технологической платформы (МИТП) создания и управления распределенной средой облачных вычислений CLAVIRE, разрабатываемой в рамках проекта «Создание распределенной вычислительной среды на базе облачной архитектуры для построения и эксплуатации высокопроизводительных композитных приложений» (Договор № 21057 от 15 июля 2010 г., шифр 2010-218-01-209) в рамках реализации постановления Правительства РФ № 218 «О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства». Программный компонент предназначен реализации лексического, синтаксического и семантического анализа скрипта на предметно-ориентированном (Domain Specific Language) языке EasyFlow, предназначенном для представления цепочек заданий (workflow) в рамках МИТП CLAVIRE.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Программный компонент разбора скрипта EasyFlow CLAVIRE/EasyFlow RU.СНАБ.80066-06 01 19 предназначен для обеспечения лексического, синтаксического и семантического анализа скрипта на языке EasyFlow (специализированном языке представления WF, разрабатываемом в рамках проекта) с целью получения его внутреннего представления в виде объектного дерева.

Компонент представляет собой программную библиотеку, предназначенную для исполнения в средах Microsoft .NET 4.0 и Miscrosoft Silverlight 4.0, разработанную на языке C#. Для обеспечения лексического и синтаксического разбора в нем используется библиотека генерации синтаксических анализаторов ANTLR.

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Функциональное назначение компонента CLAVIRE/EasyFlow состоит в обеспечении возможности работы других модулей и компонентов распределенной среды с представлением абстрактного WF, заданного в форме скрипта на DSL-языке EasyFlow.

Этот компонент обладает следующими функциональными возможностями:

1) получение скрипта EasyFlow в виде файла, потока или строки;
2) лексический анализ полученного скрипта и обнаружение ошибок на уровне лексем, а также представление скрипта в виде потока лексем;
3) синтаксический анализ на основе полученного потока лексем, состоящий в построении внутреннего представления, и обнаружение синтаксических ошибок с сообщением развернутой информации о них;
4) семантический анализ скрипта с целью построения связей между различными блоками WF, а также с целью выявления логических ошибок;
5) сбор информации о соответствии отдельных элементов внутреннего представления WF его текстовому представлению (номера строки и столбца в файле и участки текста, соответствующие тем или иным элементам внутреннего представления);
6) предоставление в результате разбора внутреннего представления WF в виде объектного дерева, пригодного для дальнейшей обработки другими компонентами.

Таким образом, описываемый компонент являет собой основу для реализации возможности задания пользователем абстрактного WF в виде декларативно-императивного описания с использованием терминов предметной области.

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

Общая функционально-логическая схема компонента представлена на рис. 3.1.

Рисунок 3.1 – Структурная схема МРС EasyFlow

Как видно из рисунка, компонент состоит из трех анализаторов кода (лексического, синтаксического и семантического). Лексический анализатор предназначен для разбиения исходного текста EasyFlow на набор лексем, синтаксический на основе этого набора анализирует язык, используя формальное описание его грамматики, а семантический – производит действия, связанные с определением сути описанных в скрипте действий. Модуль интерпретации на основе формального представления WF, полученного от блока анализа кода, производит запуск WF с помощью подсистемы исполнения.

Взаимосвязь функциональной схемы работы компонента EasyFlow с другими компонентами и модулями в рамках распределенной среды представлена на рис. 3.2, из которого видно, что обработка скрипта представляет собой процесс последовательных трансформаций и попутной обработки скрипта с построением внутреннего представления.

Скрипт поступает в компонент формирования и исполнения WF от пользовательского интерфейса в текстовом виде для первичного лексического анализа. На этом этапе на основе формального определения грамматики языка EasyFlow происходит разбиение исходного текста на набор лексем, соответствующих основным понятиям языка (идентификатор, оператор, строка, число, комментарий и пр.), посредством лексического анализатора. При этом удаляются комментарии, пробелы, символы табуляции и возврата каретки. На выходе получается поток лексем с информацией об их типе и положении в тексте. На этом этапе происходит также первичная обработка ошибок, связанная с

нахождением в тексте неприемлемых символов и лексем, не соответствующих формальной грамматике языка (см. Приложение 1).

Рисунок 3.2 – Функциональная схема работы компонента формирования и исполнения WF

Полученный набор лексем служит исходными данными для работы следующего блока – синтаксического анализатора, который на основе формальной грамматики языка выделяет из текста отдельные конструкции и строит внутреннее представление скрипта в объектном виде. На этом этапе графический пользовательский интерфейс может получать информацию о графическом представлении WF, данные о возникающих ошибках, а также служебную информацию для подсветки синтаксиса в текстовом редакторе и осуществления контекстных подсказок.

После синтаксического анализа происходит вызов блока семантического анализа, который на основе знаний, получаемых из специализированного хранилища знаний о пакетах (компонент-база пакетов), определяет состав вызываемых пакетов, их формат,

набор параметров, а также проверяет возможность исполнения переданного WF и его непротиворечивость (например, отсутствие циклических зависимостей или соответствие значений параметров допустимым диапазонам). На выходе этого этапа, в случае отсутствия критических ошибок, получается внутреннее представление WF, пригодное для передачи в компонент запуска.

Пример разбора текста скрипта во внутреннее представление WF показан на рис. 3.3 (слева стрелками показаны зависимости данных от запусков).

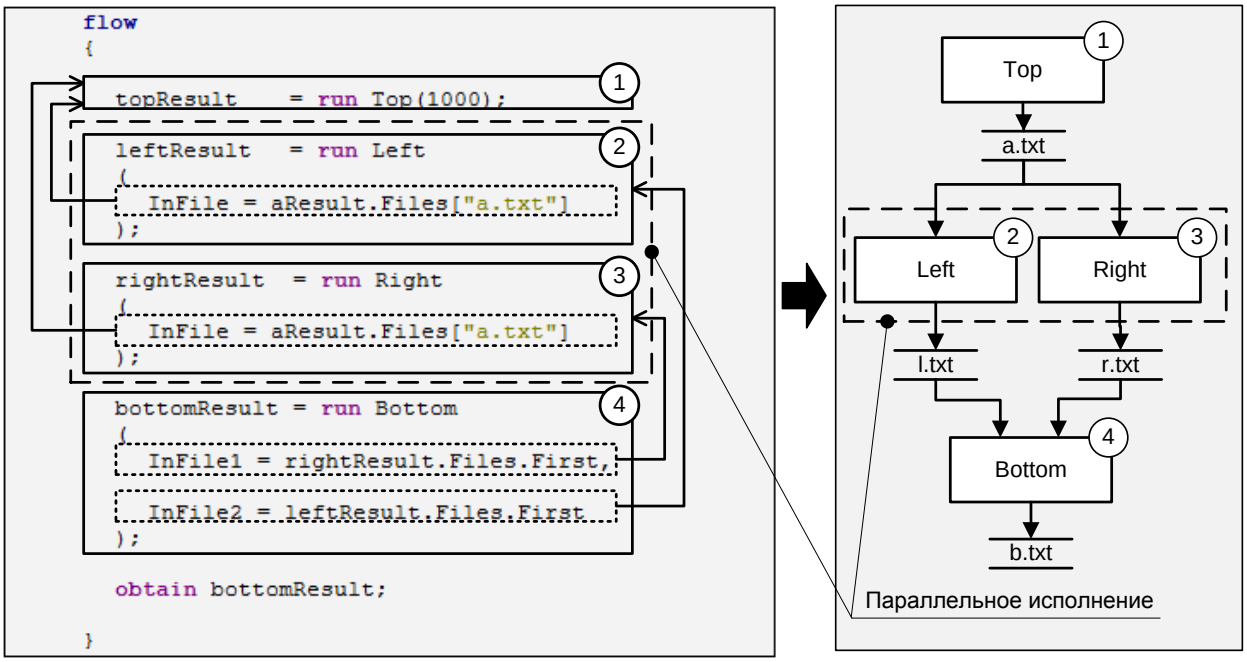

Рисунок 3.3 – Пример разбора скрипта EasyFlow

Из рисунка видно, что семантический анализатор учитывает зависимости по данным и строит соответствующее скрипту дерево исполнения. При этом учитывается возможность параллельного исполнения нескольких блоков в случае, если они не зависят друг от друга и находятся на одном уровне (им в одно и то же время становятся доступными все необходимые входные данные).

Для организации внутреннего представления WF используется набор классов, каждый из которых соответствует конкретному элементу языка (идентификатор, описание шага и т.п.). Структура этих классов представлена на рис. 3.4.

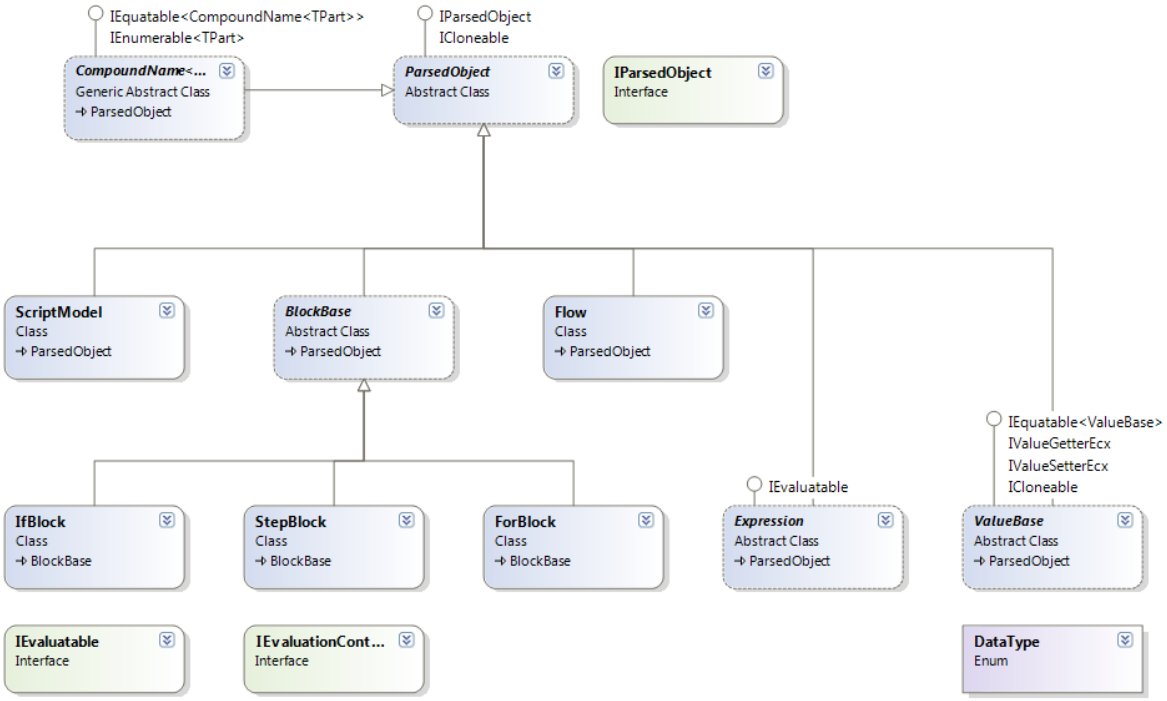

Рисунок 3.4 – Частичная структура классов внутреннего представления WF

В библиотеку также включен набор классов для обеспечения системы типов. Набор доступных для использования типов данных следующий: File, String, Int, Double, Bool, List, Enum.

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Все блоки компонента разбора скрипта EasyFlow реализованы на основе платформы Microsoft .NET версии 4.0. Языком разработки является C# версии 3.0. Сам компонент реализован в виде сборок .NET (EasyFlow.dll) и Silvelright (EasyFlow.Silverligh.dll), которые предоставляют необходимые интерфейсы и классы.

При разработке лексического и синтаксического анализатора использован генератор парсеров грамматик ANTLR версии 3.0 со средой исполнения, предназначенной для выполнения на платформе .NET и Silverlight.

Компонент предназначен для использования в рамках платформы .NET и Silverlight (в ее стандартной реализации для ОС Windows или для реализации Mono/Moonlight для ОС Linux/UNIX). Для работы компонента требуются следующие внешние библиотеки: NLog.dll (библиотека журналирования), Antlr3.Runtime.dll и antlr.runtime (библиотека времени исполнения ANTLR).

Требования к аппаратному обеспечению, на котором будет устанавливаться компонент: процессор x86 или совместимый, объем оперативной памяти – не менее 256 МБ, объем свободного места на жестком диске – не менее 100 МБ.

5. ВЫЗОВ И ЗАГРУЗКА

Компонент не требует особой установки на целевую систему, так как представляет собой динамическую сборку .NET, достаточную для простого копирования библиотеки в папку с использующим его проектом. Для применения функциональных возможностей компонента необходимо подключить библиотеку EasyFlow.dll (или EasyFlow.Silverlight.dll в случае использования с Silverlight) к использующему его приложению и инстанциировать содержащиеся в ней классы.

Для использования компонента требуется написать код, подобный приведенному в листинге 5.1.

Листинг 5.1. Пример вызова компонента разбора скрипта EasyFlow

```
ParserSettings settings = new ParserSettings
{
    CollectScopeInformation = true,
    CollectTextInformation = true,
    LogRuleTraces = true
};


ScriptParser parser = new ScriptParser(settings);

Stream inStream = new FileStream("Test1.flow", FileMode.Open);

ParseResult parseResult = null;
try
{
    parseResult = parser.Parse(inStream);
}
catch (RecognitionException recExc)
{
    Console.WriteLine("Line: {0}, LinePosition: {1}, Index: {2}, Token: {3}", recExc.L
ine, recExc.CharPositionInLine, recExc.Index, recExc.Token);
}
```

6. ВХОДНЫЕ ДАННЫЕ

Входными данными для компонента являются настройки парсинга (в виде объекта класса *ParserSettings*) и скрипт EasyFlow, который может быть передан как:

- текстовый файл, содержащий скрипт;
- строка (String), хранящая содержимое скрипта;

- поток .NET (Stream), ссылающийся на любого поставщика данных (файл, строка, участок памяти, сетевое соединение и др.).

7. ВЫХОДНЫЕ ДАННЫЕ

Выходными данными для компонента является объект класса ScriptModel, содержащий следующую информацию:

- граф внутреннего представления WF в виде объекта класса Flow;
- список требуемых файлов для запуска WF, описанных в секции require.

ПРИЛОЖЕНИЕ 1. ЯЗЫК ОПИСАНИЯ КОМПОЗИТНЫХ ПРИЛОЖЕНИЙ В ФОРМЕ WF

Язык описания композитных приложений в форме WF EasyFlow (далее – Язык или EasyFlow) предназначен для описания инструкций по выполнению композитных приложений в распределенной вычислительной среде. Он предоставляет конечному пользователю гибкие возможности по заданию различных форм потоков вычислений, в рамках которых происходят выполнение различных прикладных пакетов, генерация выходных данных, их получение, конвертация и обработка.

Основной целью Языка является полное абстрагирование от особенностей распределенной вычислительной среды, в которой работает пользователь. Это делает описание задания полностью независимым от той вычислительной среды, которая доступна на данный момент. Такой подход можно кратко выразить так: EasyFlow – это высокоуровневый язык описания *абстрактных* workflow (AWF). Такой подход позволяет описывать саму решаемую задачу, а не способ ее исполнения на конкретной вычислительной архитектуре.

П1.1. Основные элементы языка

В данном разделе описаны элементы языка EasyFlow с использованием нотации ANTLR (см. Приложение 2). Полная грамматика EasyFlow в нотации ANTLR приведена в Приложении 3.

Язык EasyFlow имеет следующие основные характеристики:

- скрипт представляет собой текстовый файл (с расширением .flow) в кодировке UTF-8;
- EasyFlow является языком, зависящим от регистра (т.е. myVariable и MyVariable представляют собой различные идентификаторы);
- типизация переменных – динамическая, слабая, с ограниченными возможностями по указанию типов;
- является платформо-независимым (с точки зрения различия операционных систем и вычислительных платформ);
- является интерпретируемым.

Переводы строк и пробелы. В EasyFlow переводы строк и пробелы являются разделителями различных конструкций языка. Для определения работы с переводами строк и пробелами в грамматику языка внесены правила, приведенные в листинге П1.1.

Листинг П1.1. Правила для обработки пробельных символов

```
wsn
  : WS
  | NEWLINE
  ;

wsnp
  : wsn+
  ;

wsns
  : wsn*
  ;

WS
  : ' ' | '\t' ;

NEWLINE
  : '\r'? '\n'
  ;
```

Пробельными символами считаются: перевод строки, пробел и знак табуляции. Переводы строк могут быть следующими: «\r\n» (Windows-style) и «\n» (Unix-style).

Для удобства определены правила wsns (любое количество пробельных символов) и wsnp (один и более пробельных символов), которые активно используются во всей грамматике языка.

Комментарии. Всюду в тексте скрипта могут встречаться однострочные комментарии после символов «//» и до конца строки, которые игнорируются парсером. Многострочные комментарии заключаются внутри символов «/*» и «*/».

Служебные символы – «{», «}», «[», «]», «(», «)», «=», «;» (точка с запятой), «:» (двоеточие), «.» (точка), «,» (запятая), «~» (тильда), «<-» (стрелка влево). Определение терминалов для этих символов приведено в листинге П1.2.

Листинг П1.2. Определение терминалов для служебных символов

```
LCUR
    : '{'
    ;

RCUR
    : '}'
    ;

LSQ
    : '['
    ;
RSQ
    : ']'
```

```
    ;

LPAREN
    : '('
    ;

RPAREN
    : ')'
    ;

ASSIGN
    : '='
    ;

SEMICOLON
    : ';'
    ;

COLON
    : ':'
    ;
DOT
    : '.'
    ;

COMMA
    : ','
    ;

TILDA
    : '~'
    ;

ARROW
    : '<-'
    ;
```

Идентификаторы языка EasyFlow определяются правилами, приведенными в листинге П1.3. Они могут начинаться буквой английского алфавита или знаком подчеркивания и содержать любое количество букв английского алфавита, цифр и знаков подчеркивания.

Листинг П1.3. Определение идентификаторов

```
ID
    : ID_BASE
    ;

fragment ID_BASE
    : ('a'..'z'|'A'..'Z'|'_' ) ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
    ;
```

Идентификаторы могут быть простыми (ID) и для указания констант (ID_CONST), которые префиксируются символом «@».

Строки в EasyFlow – это последовательности символов, заключенные в двойные кавычки вида «"». Внутри кавычек могут располагаться как обычные символы, так и escape-последовательности символов для ввода специальных символов:

- «\"» – символ «"» (двойные кавычки);
- «\b» – символ backspace;

- «\t» – символ табуляции;
- «\n» – символ перевода строки;
- «\f» – символ разрыва раздела;
- «\r» – символ возврата каретки;
- «\\»– символ «\»;
- «\'» – символ «'» (одинарные кавычки).

Строки могут содержать коды символов в шестнадцатеричном (вида «\uXXXX», где X – шестнадцатеричная цифра) и восьмеричном (вида «\XXX», где X – восьмеричная цифра) представлениях.

Листинг П1.4. Определение терминалов для строк

```
STRING
    :  '"' ( (~('\\' | '"'))| ESC_SEQ)* '"'
    ;

fragment HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment UNICODE_ESC
  : '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
  ;

fragment RULETTER
  : '\u0400'..'\u04FF'
```

Целые числа в EasyFlow представляются стандартным образом в десятичной системе счисления и могут иметь знак. Определение целых чисел приведено в листинге П1.5.

Листинг П1.5. Определение терминалов для целых чисел

```
INT : ('+' | '-')? DEC_DIGIT+
    ;

fragment DEC_DIGIT : ('0'..'9') ;
```

Дробные числа в EasyFlow представляются стандартным образом в десятичной системе счисления и могут иметь знак, целую часть, мантиссу и показатель степени

(примеры: 0.14, 0.15e+14, .34e10, 5.e–14). Определение целых чисел приведено в листинге П1.6.

Листинг П1.6. Определение терминалов для дробных чисел

```
DOUBLE
    :   DEC_DIGIT+ '.' DEC_DIGIT* EXPONENT?
    |   '.' DEC_DIGIT+ EXPONENT?
    |   DEC_DIGIT+ EXPONENT
    ;

fragment DEC_DIGIT : ('0'..'9') ;

fragment EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;
```

Булевы константы в EasyFlow представляют собой ключевые слова: true (истина) и false (ложь). Определение булевых констант приведено в листинге П1.7.

Листинг П1.7. Определение терминалов булевых констант

```
BOOL
    :  ('true') | ('false')
    ;
```

Выражения в EasyFlow в текущей версии являются простыми и могут представлять собой:

- строку;
- список (см. ниже);
- целое число;
- дробное число;
- булеву константу;
- константный идентификатор (ID_CONST, см. выше);
- выражение доступа (см. ниже).

Определение правила для выражения приведено в листинге П1.8.

Листинг П1.8. Определение правила для выражения

```
expression
  : STRING
  | INT
  | DOUBLE
  | BOOL
  | list
  | ID_CONST
  | varIdentifier
  ;
```

Выражения доступа. В EasyFlow для описания доступа к различным сложным объектам (именам пакетов, структурам) используются выражения доступа (varIdentifier),

которые представляют собой идентификаторы, разделенные точками и, возможно, имеющие индексаторы, заключенные в квадратные скобки (примеры: object.Field, object.Array[13], ORCA.List[“Hello”].str). Определение правил для выражений доступа приведено в листинге П1.9.

Листинг П1.9. Определение правил для выражения доступа

```
varIdentifier
  : simpleVarIdentifier( wsns DOT wsns simpleVarIdentifier )*
  ;

simpleVarIdentifier
  : ID (wsns varIndexer)?
  ;

varIndexer
  : LSQ wsns expression wsns RSQ
  ;
```

Списки в EasyFlow представляют собой последовательности выражений, разделенных запятыми и заключенных в квадратные скобки (примеры: [1, 2, 3], [a, [c, d]]). Списки могут быть вложенными и содержать значения различных типов. Определение правил для списков приведено в листинге П1.10.

Листинг П1.10. Определение правил для списков

```
list
  : LSQ wsns expressionList? RSQ
  ;

expressionList
  : expression wsns
  (COMMA wsns expression wsns)*
  ;
```

Ключевые слова. EasyFlow для служебных целей резервирует следующие ключевые слова:

- **flow** – для префиксирования атрибутов WF (см. ниже);
- **require** – для определения требуемых входных файлов;
- **step** – описание начала шага;
- **~step** – описание долгоживущего шага;
- **sweep** – оператор перебора по значениям;
- **runs** – дескриптор запускаемого пакета;
- **after** – указатель явной передачи управления между шагами;
- **on** – указатель передачи управления между шагами по событиям;

- **true / false** – см. выше.
- **pre, code, post –** определение секций пре- и постобработки.

Определение терминалов для ключевых слов приведено в листинге П1.11.

Листинг П1.11. Определение терминалов для ключевых слов

```
FLOW
    : 'flow'
    ;

REQUIRE
    : 'require'
    ;

STEP
    : 'step'
    ;

SWEEP
    : 'sweep'
    ;

RUNS
    : 'runs'
    ;

AFTER
    : 'after'
    ;

ON
    : 'on'
    ;

BOOL
    :  ('true') | ('false')
    ;

POST
  : 'post';

fragment CODE
  : 'code';

PRE
  : 'pre';

fragment END
  : 'end';
```

Предопределенные константы. EasyFlow может использовать предопределенные средой константы (листинг П1.12). Для этого используются идентификаторы с префиксом в виде символа «@».

Листинг П1.12. Определение терминала для предопределенных констант

```
ID_CONST
    : '@' ID_BASE
    ;
```

Строение тела скрипта. Тело скрипта (правило program) представляет собой набор выражений верхнего уровня (правило topClause) – см. листинг П1.13.

Листинг П1.13. Правила program и topClause

```
wsn
   : WS
   | NEWLINE
   ;

wsnp
   : wsn+
   ;

wsns
   : wsn*
   ;

program
   : wsns ( topClause wsns )*
   ;


topClause
   : flowAttributeClause
   | stepClause
   | requireClause
   ;
```

Как видно из листинга, выражением верхнего уровня могут быть: атрибут WF (flowAttributeClause), определение шага (stepClause) и определение требования файла (requireClause). Об этих элементах см. ниже.

Атрибут WF представляет собой задание настройки для исполнения WF и имеет синтаксис, приведенный в листинге П1.14. Пример задания настройки для WF приведен в листинге П1.15.

Листинг П1.14. Определение правила для атрибутов WF

```
flowAttributeClause

   : LSQ wsns FLOW wsns COLON wsns ID wsns ASSIGN wsns expression RSQ
   ;
```

Листинг П1.15. Пример атрибутов WF

```
[flow:priority = @urgent]
[flow:author = “Paul McCartney”]
[flow:name = “Molecular geometry optimization”]
[flow:mode = @raw]
```

Тело скрипта может содержать любое количество атрибутов WF, заданных в форме, приведенной выше. Эти атрибуты задают либо некую информацию о WF

(например, данные об авторстве или названии WF), либо системные настройки, связанные с запуском. В данной версии языка поддерживаются следующие атрибуты:

- **name** – текстовое имя WF;
- **author** – информация об авторе;
- **description** – описательный текст для WF;
- **priority** – приоритет выполнения WF, который учитывается компонентом планирования. Возможные значения: *@low* (низкий), *@normal* (нормальный), *@high* (высокий);
- **mode** – режим работы WF. Возможные значения: *@urgent* (режим экстренного запуска), *@normal* (режим нормального запуска);
- **maxDuration –** максимальное требуемое время выполнения WF (требуется планировщику в urgent-режиме).

Директива требования файлов (require). Чтобы указать, какие файлы требуются WF, в EasyFlow введена директива *require*. Она представляет собой ключевое слово «require» и следующий за ним список разделенных запятыми идентификаторов, обозначающих файлы. В дальнейшем эти идентификаторы будут ассоциированы с конкретными файлами, использующимися в запусках. В рамках одного скрипта директива требования файлов может появляться неограниченное число раз.

Определение правила для директивы *require* приведено в листинге П1.16. Примеры директив приведены в листинге П1.17.

Листинг П1.16. Определение директивы require

```
requireClause
  : REQUIRE wsnp idList wsns SEMICOLON
  ;
idList
  : ID (wsns COMMA wsns ID)*
  ;
```

Листинг П1.17. Пример использования директивы require

```
require file1;
require jpg, png, gif;
```

Директива определения шага (step). Одной из основных функций EasyFlow является описание запусков вычислительных пакетов безотносительно к распределенной вычислительной архитектуре. Для этого используется директива определения шага *step*, пример задания правил для которой приведен в листинге П1.18.

Листинг П1.18. Определение правил для директивы *step*

```
stepClause
  :
    attributes
    (TILDA)? STEP wsnp id
    RUNS wsnp compoundName
    (wsnp AFTER wsnp idList)?  // Run conditions end
    wsns LPAREN
        wsns (APP wsns COLON wsns)? namedParametersList?
        ((EXEC wsns COLON wsns) namedParametersList?)?
    RPAREN
    (wsns 'pre' WS+ CODE_BLOCK)?
    (wsns 'post' WS+ CODE_BLOCK)?
    (wsns SEMICOLON)?
 ;

compoundName
  : ID ( wsns DOT wsns ID )*

attributes
  : (attribute wsns)*
  ;

attribute
  : LSQ wsns ID wsns ASSIGN wsns expression wsns RSQ
  ;

namedParametersList
  : namedParameter wsns
    (
    (
      ( COMMA wsns namedParameter wsns )+
      ( COMMA wsns )? // comma at the end
    )
    | (COMMA wsns)?
    )
  ;

namedParameter
  : COLON?ID wsns ASSIGN wsns (SWEEP wsns)? expression
  ;
```

Директива *step* задает параметры и связи с другими шагами для выполняемого вычислительного пакета. Полный формат описания шага включает: атрибуты шага, имя шага, запускаемый пакет, условия получения управления (директива *after*), список входных параметров и файлов, а также описание секции постобработки. Пример полного определения шага приведен в листинге П1.19.

Листинг П1.19. Пример полного определения выполнения шага

```
require file1, file2;

step AnotherStep runs EmptyPackage ();

[priority = @high]
step StepName runs Package.Method after AnotherStep
(
  inFile1 = file1,
  inFile2 = file2,
  stringInput = "some string here",
  intInput = 100,
  doubleInput = 3.14,
  sweepParam = sweep [1, 2, 3],
  listParam = [AnotherStep.outs["out.txt"]]
)
post code ruby
  i = 1
  list = StepName.Result.outs
  list.reverse
code end

~step LongRunningStep runs LRPackage
(
  inStream <- StepName.Result.outs[“output.txt”]
);
```

Данный пример показывает, что шаг может иметь атрибуты, задающие параметры запуска этого шага: *mode* и *priority*, которые аналогичны описанным в разделе об атрибутах WF и могут перекрывать их.

После атрибутов следует заголовок шага, который включает: ключевое слово *step*, имя шага (идентификатор), идентификатор запускаемого пакета и метода, директиву *after* (необязательно) и список шагов, за которыми должен следовать данный шаг.

Шаг может быть помечен как долгоживущий с помощью символа «тильда» («~»), поставленного перед ключевым словом *step* (без пробелов). В этом случае в перечне параметров шага становится возможным указывать перенаправление потоков с помощью оператора «стрелка» («<-»).

После заголовка шага следует перечень параметров, заключенных в круглые скобки. Каждый параметр представляет собой наименование параметра и его значение, отделенное от него знаком «=». В правой части может стоять любое выражение (expression). Там может находиться директива *sweep*, которая «размножает» данный шаг, подставляя для каждого экземпляра значение из списка, следующего за директивой.

Файл скрипта EasyFlow может содержать любое количество определений запусков вычислительных пакетов. В следующем разделе описаны способы структурирования программы на языке EasyFlow, а именно особенности разработки WF, определение зависимостей по управлению и по данным, а также процедура перебора параметров (parameter sweep).

П1.2. Способы структурирования программы

Выше был описан синтаксис языка EasyFlow. Данный раздел посвящен особенностям создания WF с его помощью, а также технике выстраивания выполнения шагов относительно друг друга. В следующих пунктах подробно разбираются различные возможности описательной стороны EasyFlow.

Номенклатура названий вычислительных пакетов. Для определения шага требуется указать тот пакет, который предполагается запустить. Для этого используется нотация составного имени пакета, которая позволяет указывать не только имя запускаемого пакета, но и конкретный модуль, метод или режим, который требуется запустить. Для этого используется запись т.н. объекта запуска, которая представляет собой строку идентификаторов, разделенных точками. Ниже приведены примеры названий объектов запуска и их описания:

- ORCA.DFT – пакет ORCA в режиме работы по методу DFT;
- ScienceVis.ThreeDGraph.Mesh – запуск научного визуализатора, вызов его модуля ThreeDGraph и построение графика методом Mesh.

Такая нотация позволяет очень гибко описывать встраиваемые в комплекс пакеты и делать вызов отдельных часто используемых режимов и модулей более прозрачным.

Порядок передачи управления между шагами. Так как WF представляет собой ориентированный граф, в EasyFlow введены возможности по организации его структуры, а именно механизмы определения порядка выполнения шагов. Они делятся на два вида: зависимости по управлению и зависимости по данным.

Зависимости по управлению представляют собой явные указания на то, что шаг A должен начать свое исполнение после шага B с помощью директивы *after* (листинг П1.20).

Листинг П1.20. Пример зависимостей по управлению

```
step A1 runs Pkg0 ();
step A2 runs Pkg1 ();
step B runs Pkg2 after A2 ();
step C runs Pkg3 after A2 ();
step D runs Pkg4 after B, C, A1 ();
```

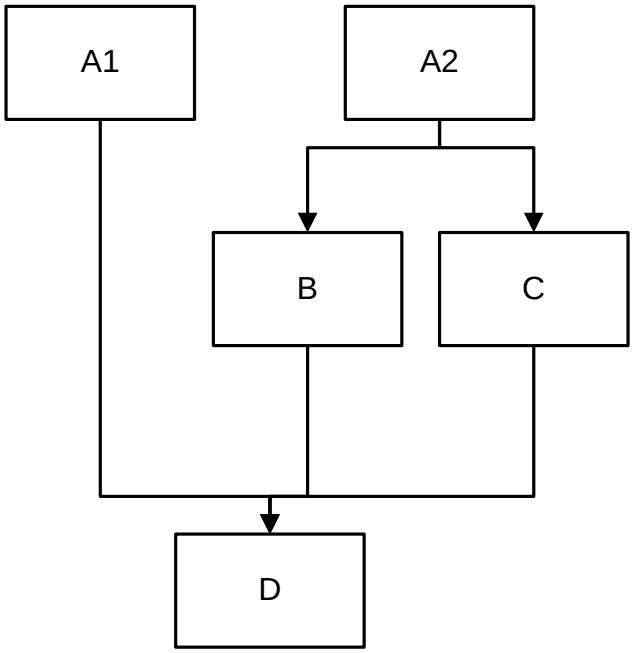

Рисунок П1.1 – Иллюстрация к листингу П1.20

Как видно из листинга, шаги A1 и A2 выполняются первыми, так как не зависят ни от каких других шагов. Шаги B и C выполняются после шага A1, а шаг D выполняется по завершении шагов B, C и A2. Эта запись может быть представлена в виде графа на рис. П1.1.

Зависимости по данным представляют собой неявные указания на зависимости между шагами, которые анализируются при интерпретации скрипта EasyFlow. Данный тип зависимости определяет порядок выполнения шагов: зависимые шаги могут быть запущены только после готовности всех требуемых для их работы данных. Такие зависимости могут присутствовать в описываемом WF одновременно с зависимостями по управлению, что позволяет очень гибко настраивать порядок выполнения шагов. Пример зависимостей по данным приведен в листинге П1.21, а соответствующий этому описанию граф – на рис. П1.2.

Листинг П1.21. Пример зависимостей по данным и управлению

```
step A1 runs Pkg0 ();

step A2 runs Pkg1 ();

step B runs Pkg2
(
  inFile = A1.outs["out.txt"]
);

step C runs Pkg3 after A2 ();

step D runs Pkg4 after C, A2 ();
```

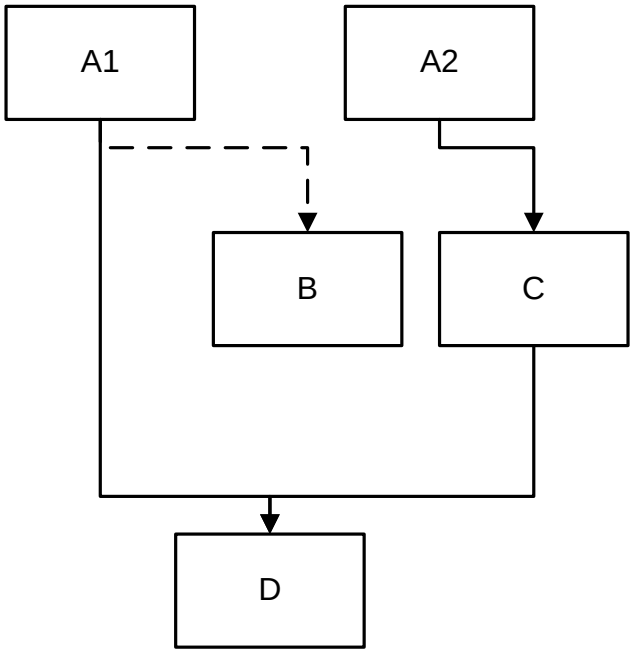

Рис. П1.2. Иллюстрация к листингу П1.21

Данный пример отличается от листинга П1.20 тем, что в описании скрипта шаг C зависит от шага A2 по данным, т.к. явно использует его результаты (A2.outs["out.txt"]).

Автоматический перебор параметров. Для удобства пользователя при проведении экспериментов в EasyFlow введена возможность автоматического варьирования параметров (parameter sweep). Такая задача часто возникает, когда необходимо запустить один и тот же вычислительный пакет, закрепив одни и варьируя другие параметры. Для этого в язык введена директива *sweep*, которая принимает список параметров для варьирования и из одного шага делает *N* шагов. Здесь *N* – число элементов в декартовом произведении списков варьирования для различных параметров.

Пример варьирования по двум параметрам (iterations и precision) представлен в листинге П1.22.

Листинг П1.22. Пример варьирования параметров

```
step SweepExample runs SomePackage
(
  width = 100,
  height = 200,
  precision = [0.1, 0.01]
  iterations = sweep [100, 200, 300]
);
```

В данном примере требуется запустить пакет SomePackage, варьируя два параметра (iterations и precision) и закрепив два других (width и height). При запуске будет создано

шесть шагов с наборами пар (precision, iterations): (0.1, 100), (0.1, 200), (0.1, 300), (0.01, 100), (0.01, 200), (0.01, 300).

ПРИЛОЖЕНИЕ 2. НОТАЦИЯ ANTLR ДЛЯ ЗАДАНИЯ ФОРМАЛЬНЫХ ГРАММАТИК ЯЗЫКОВ

ANTLR представляет собой автоматический генератор лексических и синтаксических анализаторов для различных языков на основе их формальных грамматик в форме, близкой к расширенной форме Бэкуса-Наура. Для описания формальных грамматик в нем предусмотрен простой синтаксис, описание которого приведено ниже.

Описание грамматики ANTLR представляет собой текстовый файл в кодировке UTF-8, содержащий определения правил и терминальных символов описываемого языка.

Листинг П2.1. Пример грамматики

```
grammar SampleGrammar;

rule
  : TERMINAL +
  ;

TERMINAL
  : 'hello'
  ;
```

В листинге П2.1 приведен пример простейшей грамматики, состоящей из одного правила и одного терминала. В начале файла должно стоять ключевое слово grammar и следующее за ним название грамматики языка, обычно совпадающее с названием самого языка. Как было указано, для составления формальной грамматики применяются описания двух видов: терминалы и правила.

Терминалы представляют собой указания для построения лексического анализатора и служат для первичного разбиения текста на отдельные лексемы. В нотации ANTLR терминалы именуются только заглавными буквами. Терминал описывается так: название терминала, двоеточие, подстановка для терминала (в примере – это строка 'hello', что означает, что TERMINAL определяет все строковые вхождения слова hello в тексте программы), точка с запятой, указывающая на окончание определения.

Для описания языка в нем определяются все терминальные символы (ключевые слова, строки, числа, идентификаторы и пр.), т.е. атомарные базовые единицы языка. На основании описаний терминальных символов строится лексический анализатор, который преобразует текст программы в поток разбитых по правилам лексем.

После описания терминалов языка описываются *правила*. Они именуются со строчной буквы и представляют собой указания для построения нисходящего рекурсивного синтаксического анализатора LL(*). Совокупность правил задает описание

грамматики языка. Правила описываются так: название правила, точка с запятой, подстановка для правила (в примере листинга П2.1 – это строка «TERMINAL+»), точка с запятой, указывающая на окончание определения.

Правила могут рекурсивно ссылаться на другие правила и на себя, формируя структуру формальной грамматики.

Для задания подстановок для терминалов и правил существует набор операторов и квантификаторов, которые указывают на порядок появления терминалов и правил, на их возможное или требуемое количество, присутствие. Также используются скобки для группировки сложных выражений.

В листинге П2.2 приведены все возможные операторы и квантификаторы, а ниже дано их описание. Все операторы и квантификаторы могут применяться для описания как терминалов, так и правил.

Листинг П2.2. Пример грамматики с операторами и квантификаторами ANTLR

```
grammar MyLang;

main_rule
  : BEGIN
      ((assign | function_call) ';')+
    END
  ;

function_call
  : IDENTIFIER '(' (expression ',')* ')'
  ;

assign
  : IDENTIFIER '=' expression
  ;

expression
  : IDENTIFIER
  | NUMBER
  | function_call
  | expression operator expression
  | '(' expression ')'
  ;

operator
  : '+' | '*' | '-' | '\\'
  ;

number
  : INTEGER | DOUBLE
  ;

INTEGER
  : ('0'..'9')+
  ;

DOUBLE
  : INTEGER ('.' INTEGER)?
  ;

IDENTIFIER
  : ('a'..'z' | '_' ) ('a'..'z')*
  ;

BEGIN
  : 'begin'
  ;
```

```
END
  : 'end'
  ;
```

Листинг П2.2 представляет собой описание грамматики простейшего языка, который поддерживает операции присваивания и вызовы функций. В операциях присваивания могут использоваться арифметические выражения. Текст программы обрамляется ключевыми словами «begin» и «end». Начнем рассмотрение описания этой грамматики с терминальных символов.

Терминал INTEGER задает определение для целых чисел с помощью квантификатора «+», который означает, что элемент, за которым он следует, должен повториться один или более раз. В данном случае в качестве аргумента этого квантификатора служат цифры от нуля до девяти. Таким образом, целое число задается как подряд идущие одна или более цифр.

На основе терминала INTEGER определяется терминал DOUBLE, задающий дробное число. В нем используются группирование скобок и квантификатор «?», означающий, что элемент, за которым он следует, должен появиться в тексте один раз или ни разу, таким образом, дробное число – это целое число и, возможно, следующая за ним точка и целое число, обозначающее мантиссу.

Для определения терминала IDENTIFIER (идентификатор) используются скобки и квантификатор «*», означающий, что элемент, за которым он следует, может любое количество раз. Таким образом, идентификатор начинается с английской буквы в нижнем регистре или знака подчеркивания. Далее может следовать любое количество (в том числе и нулевое) английских букв в нижнем регистре.

Для описания любого языка требуется описать главное правило, с которого синтаксический анализатор будет начинать разбор. В листинге это правило main_rule, задающее тело программы. В нем используются определения как терминалов, так и других правил. Это правило описывает программу, которая начинается с терминала BEGIN и заканчивается терминалом END. Между ними могут помещаться, как видно из определения, одно или более следующих утверждений, заданных собственными правилами: assign (оператор присваивания) или function_call (вызов функции), за которыми должна следовать точка с запятой.

Вызов функции, как видно из листинга, – это идентификатор и список из нескольких (или ни одного) выражений (правило expression), заключенных в скобки и разделенных запятой.

Оператор присваивания – это идентификатор, знак равенства и присваиваемое выражение.

В определении правила для выражения (expression) используются два новых элемента: выбор варианта («|») и рекурсивный вызов, т.е. вызов правила из самого себя. Это правило может быть прочитано так: выражение – это:

- либо число (целое или дробное);
- либо идентификатор;
- либо вызов функции;
- либо оператор с двумя выражениями;
- либо выражение, заключенное в скобки.

Таким образом, например, строка 45 * (3 + (14 – 56)) последовательно разберется так:

Листинг П2.3. Последовательность разбора выражения

```
expression (45) operator (*) expression ( (3 + (14 – 56)) );
NUMBER (45) operator (*) expression ( (3 + (14 – 56)) )
NUMBER (45) operator (*) ‘(’ expression ‘)’
NUMBER (45) operator (*) ‘(’ NUMBER (3) operator expression ‘)’
NUMBER (45) operator (*) ‘(’ NUMBER (3) operator (+) expression ‘)’
NUMBER (45) operator (*) ‘(’ NUMBER (3) operator (+)  ‘(’ expression ‘)’ ‘)’
NUMBER (45) operator (*) ‘(’ NUMBER (3) operator (+)  ‘(’ expression operator (–)
expression ‘)’ ‘)’
NUMBER (45) operator (*) ‘(’ NUMBER (3) operator (+)  ‘(’ expression operator (–)
expression ‘)’ ‘)’
NUMBER (45) operator (*) ‘(’ NUMBER (3) operator (+)  ‘(’ NUMBER (14) operator (–
)NUMBER (56) ‘)’ ‘)’
```

В листинге П2.4 приводится пример программы на языке, формальная грамматика которого задана в листинге П2.2.

Листинг П2.4. Пример программы на модельном языке

```
begin
  a = 14;
  b = 15;
  c = a * b + b;
  g = sum(sin(c), 14 / 5, cos(a));
  exit();
end
```

ПРИЛОЖЕНИЕ 3. ФОРМАЛЬНАЯ ГРАММАТИКА ЯЗЫКА EASYFLOW В НОТАЦИИ ANTLR

```
grammar EasyFlow;

program
   : wsns ( topClause wsns )*
   ;

// White space rule
wsn
   : WS
   | NEWLINE
   ;

wsnp
   : wsn+
   ;

wsns
   : wsn*
   ;

requireClause
   : REQUIRE wsnp idList wsns SEMICOLON
   ;

idList
   : ID (wsns COMMA wsns next=ID)*

   ;

topClause
   :
     flowAttributeClause
   | stepClause
   | requireClause
   ;

flowAttributeClause
   : LSQ wsns FLOW wsns COLON wsns ID wsns ASSIGN wsns expression RSQ
   ;

attribute
   : LSQ wsns ID wsns ASSIGN wsns expression wsns RSQ
   ;

attributes

   : (attribute wsns)*
   ;

stepClause
   :
     attributes (TILDA)? STEP wsnp id
     RUNS wsnp compoundName

     (wsnp AFTER wsnp idList)?  // Run conditions end
     wsns LPAREN wsns (APP wsns COLON wsns)? namedParametersList?
     ((EXEC wsns COLON wsns) execParams=namedParametersList?)?
     RPAREN
     (wsns 'post' WS+ CODE_BLOCK)?
     (wsns SEMICOLON)?
 ;

varIdentifierList
   : varIdentifier (wsns COMMA wsns varIdentifier)*
   ;

varIdentifier
   : simpleVarIdentifier ( wsns DOT wsns simpleVarIdentifier)*
   ;
```

```
simpleVarIdentifier returns [SimpleVarIdentifier SimpleVarIdentifier]
  : ID (wsns varIndexer)?
  ;

varName
  : ID
  ;

varIndexer
  : LSQ wsns expression wsns RSQ
  ;

filePointer
  : ID
  ;

compoundName
  : compoundNameComponent ( wsns DOT wsns compoundNameComponent)*
  ;

compoundNameComponent
  : ID
  ;

namedParametersList
  : namedParameter wsns
    (((COMMA wsns namedParameter wsns)+ (COMMA wsns)?) | (COMMA wsns)?)
  ;

namedParameter
  : COLON? ID wsns (ASSIGN | ARROW) wsns (SWEEP wsns)? expression
  ;

expression
  : STRING
  | INT
  | DOUBLE
  | BOOL
  | list
  | ID_CONST
  | varIdentifier
  ;

list
  : LSQ wsns expressionList? RSQ
  ;

expressionList
  : expression wsns (COMMA wsns expression wsns)*
  ;

id
  : ID
  ;

ML_COMMENT
    :   '/*' (options {greedy=false;} : .)* '*/' {$channel=HIDDEN;}
    ;

SL_COMMENT
    : '//' .* NEWLINE {$channel=HIDDEN;}
    ;

POST
  : 'post';

fragment CODE
  : 'code';

PRE
  : 'pre';

fragment END
  : 'end';

CODE_BLOCK
  : CODE
```

```
    (options {greedy=false;} : .)*
    CODE (' ' | '\t')+ END
  ;

WS
    : ' '
    | '\t'
    ;

ARROW
    : '<-'
    ;

LCUR
    : '{'
    ;

RCUR
    : '}'
    ;

LSQ
    : '['
    ;
RSQ
    : ']'
    ;

LPAREN
    : '('
    ;

RPAREN
    : ')'
    ;

ASSIGN
    : '='
    ;

SEMICOLON
    : ';'
    ;

COLON
    : ':'
    ;

FLOW
    : 'flow'
    ;

APP
    : 'app'
    ;

EXEC
    : 'exec'
    ;

REQUIRE
    : 'require'
    ;

STEP
    : 'step'
    ;

SWEEP
    : 'sweep'
    ;

RUNS
    : 'runs'
    ;

AFTER
    : 'after'
```

```
    ;

ON
    : 'on'
    ;

BOOL
    :  ('true') | ('false')
    ;

ID_CONST
    : '@' ID_BASE
    ;

ID
    : ID_BASE
    ;

fragment ID_BASE
    : ('a'..'z'|'A'..'Z'|'_' ) ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
    ;

STRING
    :  '"' ( (~('\\' | '"'))| ESC_SEQ)* '"'
    ;

DOUBLE
    :   ('+' | '-')? DEC_DIGIT+ '.' DEC_DIGIT* EXPONENT?
    |   ('+' | '-')? '.' DEC_DIGIT+ EXPONENT?
    |   ('+' | '-')? DEC_DIGIT+ EXPONENT
    ;

INT : ('+' | '-')? DEC_DIGIT+
    ;

fragment DEC_DIGIT : ('0'..'9') ;

fragment EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

fragment HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment UNICODE_ESC
  : '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
  ;

fragment RULETTER
  : '\u0400'..'\u04FF'
  ;

DOT
    : '.'
    ;

COMMA
    : ','
    ;

TILDA
  : '~'
  ;
NEWLINE
  : '\r'? '\n';
```

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

AW	Абстрактный workflow
DFT	Метод функционала плотности
DSL	Domain Specific Language
WF	Workflow (поток заданий)
МИТП	Многопрофильная инструментально-технологическая платформа

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	изме-ненных	заме-ненных	но-вых	Аннули-рован-ных					