электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.436.4

Синтаксис формального текстового описания UML диаграмм классов

***Сиромаха Р.В.**, студент*
кафедра «Программное обеспечение ЭВМ и информационные технологии»,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана

Научный руководитель: *Рудаков И.В., к.т.н.,*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
irudakov@bmstu.ru

Введение

UML диаграммы – это метод для описания модели информационной системы [1], который используется на этапе проектирования в процессе разработки программного обеспечения. UML диаграммы классов – это метод для описания структуры модели объектно-ориентированной программы. Этот метод основан на использовании графического описания. Существуют ситуации, когда такой подход является недостатком, потому что любое изменение, например добавление нового элемента или связи, может вызвать необходимость изменить местоположение всех элементов диаграммы для более удобной компоновки с точки зрения восприятия. Возникает необходимость составить диаграмму таким образом, чтобы она представляла собой единое целое с учетом возможности изменить или отобразить отдельную ее часть, не изменяя другие части диаграммы. Графический способ описания для этого не подходит. Необходимо разработать другой способ описания.

Одним из способов описания является использование формального языка. Это в первую очередь предполагает разработку синтаксиса формального языка, который бы позволял определить UML диаграмму классов. Для этого необходимо решить следующие задачи:

1. Выполнить классификацию элементов UML диаграммы классов.
2. Выполнить отсечение элементов, которые напрямую не влияют на представление диаграммы.
3. Составить для каждого элемента набор правил формальной грамматики.
4. Объединить набор правил в единую грамматику и определить ее тип.

Классификация элементов UML диаграммы классов

UML диаграммы имеют несколько уровней описания [2]. Нулевой уровень описывает базовые понятия. Каждый последующий уровень описывается на основе предыдущего. В итоге получается, что UML диаграмма классов описана UML диаграммой классов более низкого уровня. Каждый элемент UML диаграммы классов принадлежит некоторому классу. Для выделения этого специального понятия можно ввести термин – **UML-класс**. Таким образом, получается, что необходимо классифицировать UML-классы с точки зрения их формального текстового описания.

Все UML-классы можно разделить на абстрактные и конкретные (см. табл. 1). При этом используется обобщение свойств UML-классов за счет связи – наследования (см. рис. 1). Используется одиночное и множественное наследование.

Часть конкретных UML-классов имеют формальное текстовое описание (см. табл. 2).

Таблица 1

Классификация UML-классов по типу

Абстрактные	Конкретные
Abstraction	Association
BehavioralFeature	AssociationClass
Classifier	Class
DirectedRelationship	Comment
Element	Constraint
Feature	DataType
InstanceSpecification	Dependency
LiteralSpecification	ElementImport
MultiplicityElement	Enumeration
NamedElement	EnumerationLiteral
Namespace	Expression
PackageableElement	Generalization
Realization	GeneralizationSet
RedefinableElement	InstanceValue
Relationship	Interaface
StructuralFeature	InterafaceRealization
Type	LiteralBoolean
TypedElement	LiteralInteger
ValueSpecification	LiteralNull
	LiteralReal
	LiteralString
	LiteralUnlimitedNatural

	OpaqueExpression Operation Package PackageImport PackageMerge Parameter PrimiveType Property Slot Substitution Usage

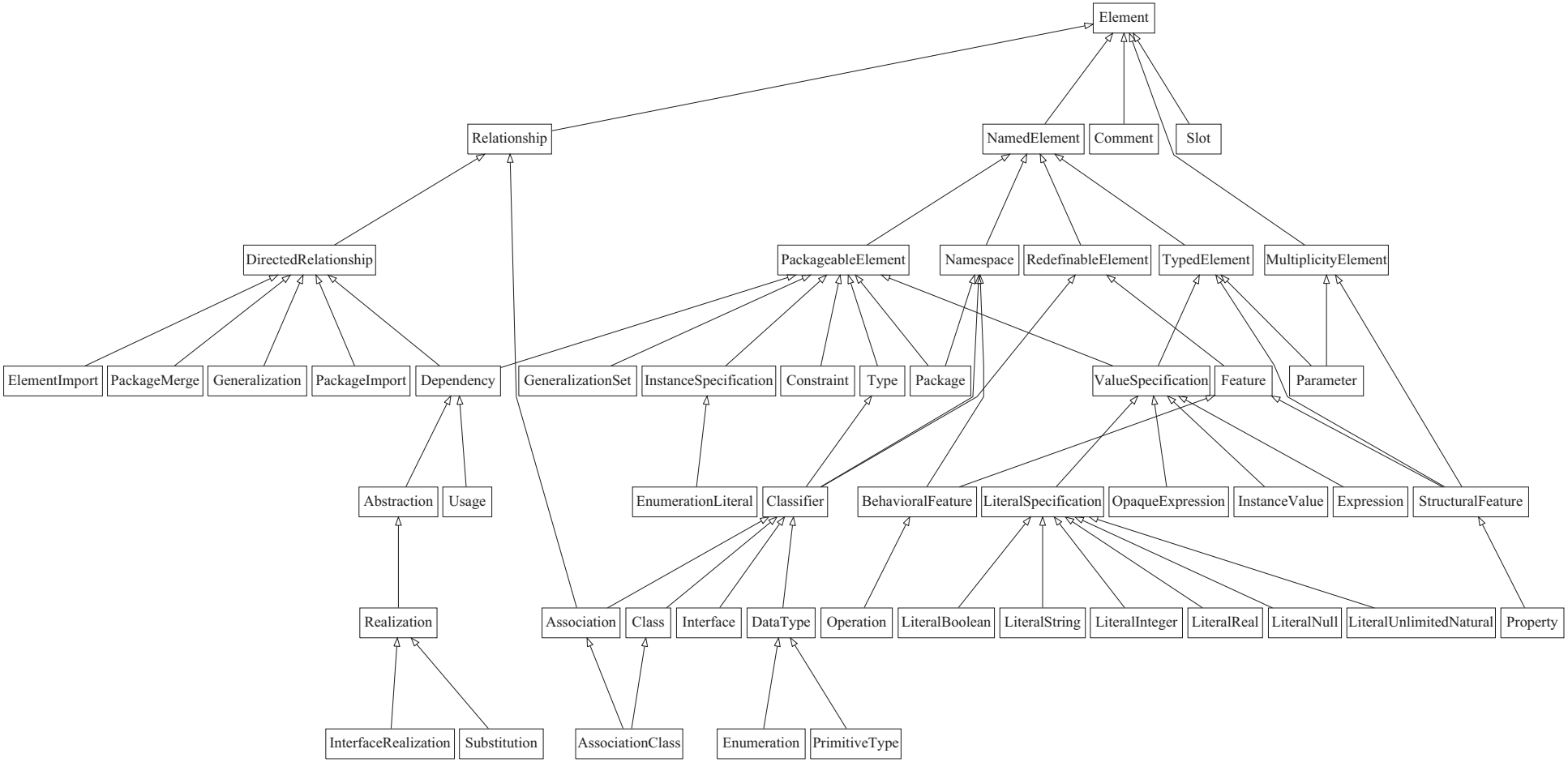

Рис. 1. Иерархия наследования UML-классов

Таблица 2

Классификация конкретных UML-классов по наличию формального текстового описания

Имеют формальное текстовое описание	Не формального текстового описания
Constraint	Association
ElementImport	AssociationClass
LiteralBoolean	Class
LiteralNull	Comment
LiteralReal	DataType
Operation	Dependency
PackageImport	Enumeration
Parameter	EnumerationLiteral
Property	Expression
	Generalization
	GeneralizationSet
	InstanceValue
	Interaface
	InterafaceRealization
	LiteralInteger
	LiteralString
	LiteralUnlimitedNatural
	OpaqueExpression
	Package
	PackageMerge
	PrimiveType
	Slot
	Substitution
	Usage

Отсечение бесполезных UML-классов

Отсечению подлежат все абстрактные UML-классы. Также отсечению подлежат такие ассоциации UML-классов, которые являются объединениями других ассоциаций (derived union). Для этого необходимо удалить связи наследования (см. рис. 2, 3). Можно составить алгоритм удаления связей наследования (см. рис. 4, 5, 6). Затем

можно отсечь бесполезные UML-классы. В результате получим набор UML-классов, каждый из которых с учетом связей можно описать набором правил грамматики.

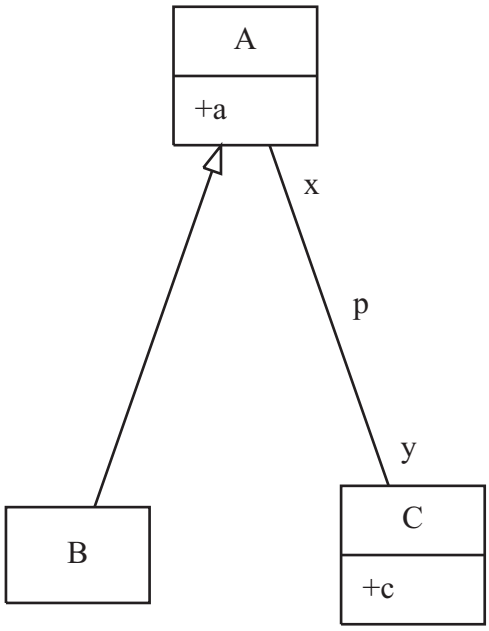

Рис. 2. Исходная UML диаграмма классов

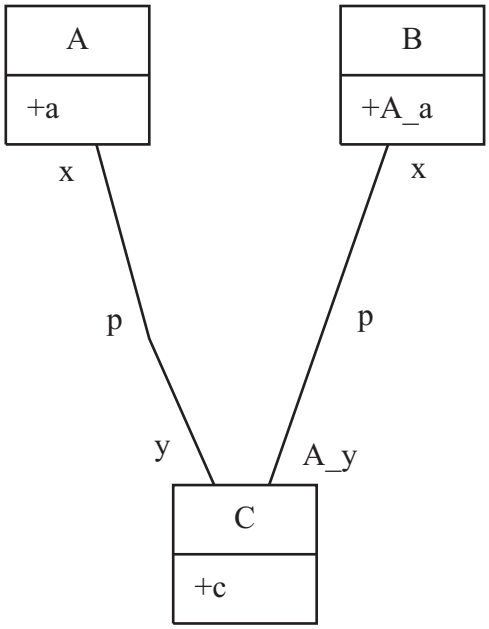

Рис. 3. UML диаграмма классов после удаления связей наследования

Рис. 4. Алгоритм удаления связей наследования

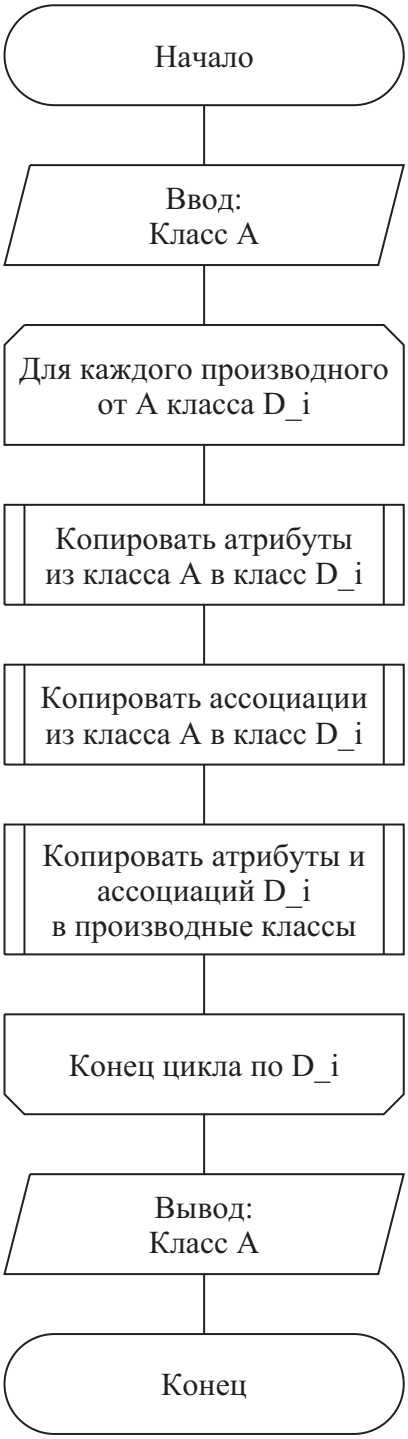

Рис. 5. Алгоритм копирования атрибутов и ассоциаций класса в производные классы

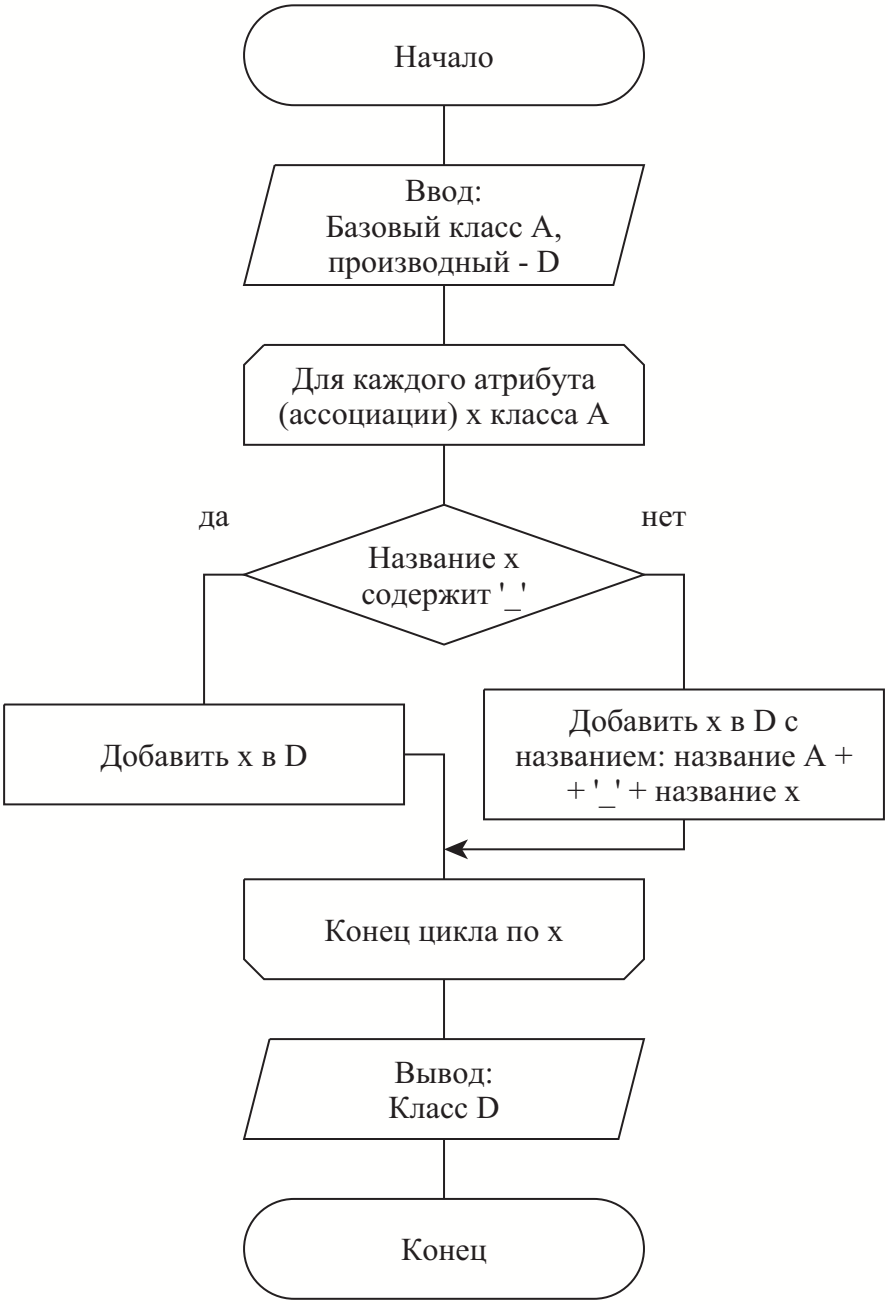

Рис 6. Алгоритм копирования атрибутов (ассоциаций) класса в производный класс

Составление правил грамматики UML-классов

Часть UML-классов имеют текстовое представление и уже описаны правилами грамматики. Эти описания можно сохранить. Для других UML-классов необходимо определить свой синтаксис, при этом можно основываться на существующих языках программирования. Главное – сохранить простоту и краткость. Описать грамматику можно с помощью расширенной формы Бэкуса-Наура [3].

Основные нетерминалы

Для начала потребуется определить нетерминалы [4] примитивных типов данных, таких как, логическое значение, целое число, вещественное число и строка. Эти нетерминалы потребуются в дальнейшем.

boolean = ’true’ | ’false’;

decimal = integer [’.’ natural];

digit = ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’;

integer = [’+’ | ’-’] natural;

name = nameValue [’::’ name];

natural = digit { digit };

real = decimal [(’e’ | ’E’) integer];

string = ’"’ stringValue ’"’;

Нетерминалы stringValue и nameValue удобнее определить в форме регулярного выражения.

stringValue = ([^"]|\"|\\)*

nameValue = \w+

Перечисления

Существуют следующие перечисления (Enumeration):

- AggregationKind -- тип агрегации свойства (Property).
- ParameterDirectionKind -- тип параметра (Parameter) операции (Operation).
- VisibilityKind -- видимость именованного элемента (NamedElement).

Для каждого перечисления и каждого литерала перечисления (EnumerationLiteral) можно ввести отдельный нетерминал.

none = ’’;

shared = ’shared’;

composite = ’composite’;

AggregationKind = none | shared | composite;

in = ’in’ | ’’;

inout = ’inout’;

out = ’out’;

ParameterDirectionKind = in | out | inout;

package = ’~’;

private = ’-’;

protected = ’#’;

public = ’+’ | ’’;

VisibilityKind = public | protected | private | package;

Атрибуты

У каждого UML-класса существуют именованные атрибуты, для каждого из которых можно выделить нетерминал, чтобы при синтаксическом анализе можно было однозначно определить, какое значение необходимо присвоить атрибуту объекта UML-класса.

aggregation = AggregationKind;

alias = string;

direction = ParameterDirectionKind;

isAbstract = ['abstract'];

isCovering = ['covering'];

isDerived = ['/'];

isDerivedUnion = ['union'];

isDisjoint = ['disjoint'];

isFinalSpecialization = ['final'];

isID = ['id'];

isLeaf = ['leaf'];

isOrder = ['ordered'];

isQuery = ['query'];

isReadOnly = ['const'];

isStatic = ['static'];

isUnique = ['unique'];

lower = LiteralInteger;

ownedComment = Comment;

upper = LiteralUnlimitedNatural;

visibility = VisibilityKind;

Нетерминал body удобнее определить в форме регулярного выражения.

body = `([^<>]|\\[<>]|\\)*`

Ассоциации

Аналогично можно поступить с ассоциациями.

classifier = type;

client = name;

contract = client;

defaultValue = value;

definingFeature = StructuralFeature;

generalization = name;

general = name { ’,’ name };

importedElement = name;

importedPackage = name;

importingNamespace = name;

instance = InstanceSpecification;

lowerValue = lower | ValueSpecification;

memberEnd = name ’::’ Property;

mergedPackage = name;

ownedAttribute = Property;

ownedLiteral = EnumerationLiteral;

ownedOperation = Operation;

ownedParameter = Parameter;

ownedParameters = ownedParameter { ’,’ ownedParameter };

packagedElement = Abstraction | Association | AssociationClass | Class | DataType | Dependency | Enumeration | GeneralizationSet | InstanceValue | Interface | Package | PrimitiveType | Substitution | Usage;

powertype = name;

receivingPackage = name;

slot = Slot;

specification = ValueSpecification;

subsettedProperty = [’subsets’ { name }];

substitutingClassifier = supplier;

supplier = name;

type = name [multplicity];

upperValue = upper | ValueSpecification;

value = ValueSpecification;

Специальные нетерминалы

Для сокращения записи можно ввести специальные нетерминалы. Также для определения класса связей-зависимостей (Dependency) можно ввести нетерминалы для определения этих связей внутри определения объектов.

abstraction = ’abstract’ supplier { ’,’ supplier } [ownedComment];

dependency = ’depend’ name supplier { ’,’ supplier } [ownedComment];

elementImport = ’element’ ’import’ importedElement [’as’ alias] [ownedComment];

multplicity = [’[’ [lowerValue ’..’] upperValue ’]’] [isOrder] [isUnique];

packageImport = ’import’ importedPackage [ownedComment];

packageMerge = ’merge’ mergedPackage [ownedComment];

substitution = ’substitute’ substitutingClassifier [ownedComment];

usage = ’use’ supplier { ’,’ supplier } [ownedComment];

Нетерминалы UML-классов

Для каждого конкретного UML-класса можно ввести нетерминал, который полностью определит класс, со всеми его атрибутами и ассоциациями.

Abstraction = ’abstraction’ [name] (client | ’{’ (client { ’,’ client }) ’}’) (supplier | ’{’ (supplier { ’,’ supplier }) ’}’) [ownedComment];

AssociationClass = isAbstract isFinalSpecialization isLeaf visibility isDerived ’association’ ’class’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | ownedAttribute | ownedOperation | memberEnd } ’}’;

Association = isAbstract isFinalSpecialization isLeaf visibility isDerived ’association’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | memberEnd } ’}’;

Class = isAbstract isFinalSpecialization isLeaf visibility ’class’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | ownedAttribute | ownedOperation } ’}’;

Comment = ’<’ body ’>’;

DataType = isAbstract isFinalSpecialization isLeaf visibility ’type’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | ownedAttribute | ownedOperation } ’}’;

Dependency = ’dependency’ name (client | ’{’ (client { ’,’ client }) ’}’) (supplier | ’{’ (supplier { ’,’ supplier }) ’}’) [ownedComment];

ElementImport = ’element’ ’import’ importingNamespace importedElement [’as’ alias] [ownedComment];

Enumeration = isAbstract isFinalSpecialization isLeaf visibility ’enum’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | ownedAttribute | ownedOperation | ownedLiteral } ’}’;

EnumerationLiteral = InstanceSpecification [ownedComment];

GeneralizationSet = isCovering isDisjoint ’generalization set’ [name ’:’] powertype [ownedComment] ’{’ generalization { ’,’ generalization } ’}’;

InstanceSpecification = visibility name [’:’ classifier [’,’ classifier]] [’=’ (’{’ [specification | slot { slot }] ’}’ | specification | slot)] [ownedComment];

InstanceValue = instance;

Interface = isAbstract isFinalSpecialization isLeaf visibility ’interface’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | ownedAttribute | ownedOperation } ’}’;

LiteralBoolean = boolean;

LiteralInteger = integer;

LiteralNull = ’null’;

LiteralReal = real;

LiteralString = string;

LiteralUnlimitedNatural = natural | ’*’;

Operation = isStatic isQuery isLeaf visibility name ’(’ [ownedParameters] ’)’ [’:’ type] [ownedComment];

PackageImport = ’import’ importingNamespace importedPackage [ownedComment];

PackageMerge = ’merge’ receivingPackage mergedPackage [ownedComment];

Package = visibility ’package’ name [ownedComment] ’{’ { packageMerge | packageImport | elementImport | packagedElement } ’}’;

Parameter = [direction] name [’:’ type] [’=’ defaultValue] [ownedComment];

PrimitiveType = isAbstract isFinalSpecialization isLeaf visibility ’primitive’ name [’:’ general] [ownedComment] ’{’ { elementImport | dependency | usage | abstraction | substitution | ownedAttribute | ownedOperation } ’}’;

Property = isStatic isReadOnly isDerivedUnion isID isLeaf subsettedProperty aggregation visibility isDerived name [’:’ type] [’=’ defaultValue] [ownedComment];

Slot = definingFeature [’=’ [value | ’{’ value { ’,’ value } ’}’]] [ownedComment];

StructuralFeature = isReadOnly name [’:’ type] | Property;

Substitution = ’substitution’ contract substitutingClassifier [ownedComment];

Usage = ’usage’ (client | ’{’ (client { ’,’ client }) ’}’) (supplier | ’{’ (supplier {’,’ supplier }) ’}’) [ownedComment];

ValueSpecification = LiteralBoolean | LiteralInteger | LiteralNull | LiteralReal | LiteralString | LiteralUnlimitedNatural | InstanceValue;

Объединение правил грамматики UML-классов

Объединение можно выполнить совмещением правил с добавлением аксиомы [4].

UML = {

Abstraction

| Association

| AssociationClass

| Class

| DataType

| Dependency

| ElementImport

| Enumeration

| GeneralizationSet

| InstanceValue

| Interface

| Package

| PackageImport

| PackageMerge

| PrimitiveType

| Substitution

| Usage

};

Полученная грамматика является контекстно-свободной.

Вывод

Разработанный синтаксис формального языка применим для описания структуры программного обеспечения при использовании объектно-ориентированной методологии программирования в терминах UML диаграммы классов, который при использовании в совокупности со всем комплексом UML диаграмм позволяет выполнять моделирование программного обеспечения.

Список литературы

1. OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1. – Август 2011.
2. OMG Unified Modeling Language (OMG UML), Infrastructure. Version 2.4.1. – Август 2011.
3. ISO/IEC 14977. – 1996.
4. Белоусов А. И., Ткачев С. Б. Дискретная математика: Учеб. для вузов / Под ред. В. С. Зарубина, А. П. Крищенко. – 3-е изд. стереотип. – М.: МГТУ им. Н. Э. Баумана 2004. – 744 с. (Сер. Математика в техническом университете; Вып. XIX). – ISBN 5-7038-1769-2.